

版权注意事项：

- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。

MANNING

jQuery in Action
Third Edition

jQuery 实战

(第三版)

[美] Bear Bibeault
Yehuda Katz 著
Aurelio De Rosa

徐雷 徐扬 译



华中科技大学出版社

<http://www.hustp.com>

jQuery 实战

(第三版)

Bear Bibeault

[美] Yehuda Katz 著

Aurelio De Rosa

徐雷 徐扬 译

华中科技大学出版社

中国·武汉



内 容 提 要

本书适合想深入学习 jQuery 的 Web 开发人员使用。jQuery 是互联网上最流行的 JavaScript 框架。本书的目标是希望读者成为 Web 高级开发人员,无论起点如何。本书深入介绍了整个 jQuery 框架,此外还专门深入讲解了插件编程,以及一些扩展开发工具和框架,比如 Bower 和 QUnit,当然还有经典的开发实战原则。每个 API 方法都使用了简明扼要的语法块来描述参数和返回值。为了便于大家理解知识,本书包含了大量的实例代码、三个插件及三个例子项目,也包含了试验网页(Lab Pages)。这些有趣的网页开发可以让大家在实战开发中快速掌握 jQuery 方法的差别,而不需要编写大量的代码。

阅读本书需要大家提前掌握 HTML、CSS 和 JavaScript 的基础编程知识。jQuery 以前的知识不是必需的,但是可以帮助大家快速理解掌握新的概念。

Original English language edition published by Manning Publications, USA. Copyright © 2015 by Manning Publications. Simplified Chinese-language edition copyright © 2016 by HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY PRESS. All rights reserved.

湖北省版权局著作权合同登记 图字:17-2016-351 号

图书在版编目(CIP)数据

jQuery 实战/(美)比尔·比博尔特(Bear Bibeault),(美)叶华达·卡兹(Yehuda Katz),(美)奥瑞里奥·罗萨(Aurelio De Rosa)著;徐雷,徐扬译. —3 版. —武汉:华中科技大学出版社,2016.7
ISBN 978-7-5680-2035-0

I. ①j… II. ①比… ②叶… ③奥… ④徐… ⑤徐… III. ①JAVA 语言-程序设计
IV. ①TP312

中国版本图书馆 CIP 数据核字(2016)第 155585 号

jQuery 实战(第三版)

(美)Bear Bibeault, Yehuda Katz, Aurelio De Rosa 著

jQuery Shizhan

徐雷 徐扬 译

策划编辑:谢燕群

责任编辑:陈元玉

责任校对:张会军

责任监印:朱 玢

出版发行:华中科技大学出版社(中国·武汉)

武昌喻家山 邮编:430074 电话:(027)81321913

录 排:华中科技大学惠友文印中心

印 刷:湖北新华印务有限公司

开 本:787mm×960mm 1/16

印 张:28.25

字 数:700 千字

版 次:2016 年 7 月第 3 版第 1 次印刷

定 价:88.00 元



华中出版

本书若有印装质量问题,请向出版社营销中心调换
全国免费服务热线:400-6679-118 竭诚为您服务
版权所有 侵权必究

《jQuery实战》好评如潮

Praise for Earlier Editions of *jQuery in Action*

每一本技术书籍都应该像这本书一样，简洁而清晰，幽默而风趣，而且回答了自己提出的所有问题。读者不会留下疑惑。

——JRoller Online Book Reviews

感谢作者的实例性行文风格！这本书是一本较全面的操作手册，可以用来作为从头开始的学习书籍，也可以作为jQuery开发者学习最佳实践经验的参考资料。

——Matthew McCullough

Denver Open Source Users Group

全面的知识体系，详细的示例代码，通俗易懂的风格，这本书就是Web开发者寻找的发掘JavaScript最大潜能的宝贵学习资源，对于任何喜欢jQuery的开发者都是必备的学习书籍。

——Michael J. Ross

Web Developer and Slashdot Contributor

卓越的写作，曼宁（Manning）出版社实战系列书籍的又一个精品系列。本书易于阅读，而且包含大量例子代码。试验页面也是学习jQuery库的绝佳途径，本书是每个Web开发者兵器库必备的武器。五星好评！

——David Sills

JavaLobby,DZone

强烈推荐学习jQuery框架本质的开发者阅读本书，它是全面发掘jQuery潜力的很好的参考书籍。

——David Hayden

MVP C#,Codebetter.com

对于任何想深入学习JavaScript高级编程，以及想编写优化且优美代码的JavaScript开发者，这都是一本强烈推荐的好书！让你远离传统JavaScript代码的困扰。

——Val's Blog

关于JavaScript的旷世经典之作！

——Joshua Heyer

Trane Inc.

湖北省版权局著作合同登记 图字 17-2014-331 号

全面的跟从体系，详细的图片片断，通俗易懂的风格，这本《就是Web开发》是网络界的一本好书。本书作者Michael J. Ross

卓敏的著作《寧曼(anning)出謀計策與系統控制中的另一個系統》(another system in the control system)一书，是其在系统控制领域的一个代表作。该书不仅是一本理论著作，也是一本实用的参考书。全书共分五章，第一章介绍系统控制的基本概念，第二章介绍系统控制的数学模型，第三章介绍系统控制的稳定性分析，第四章介绍系统控制的性能指标，第五章介绍系统控制的工程应用。本书可作为高等院校自动控制专业及相关专业的教材，也可供从事系统控制工作的工程技术人员参考。

David Tsayden
MBC & ePublisher.com

都是一本按照循序渐进的教材！本书从Java2Script入门开始，循序渐进地介绍了Java2Script的语法、数据类型、运算符、表达式、语句、函数、数组、字符串、日期、时间、窗口、事件、动画、声音、网络、数据库、XML、Web Services、AJAX、JSON、JSONP、JSONP2、JSONP3、JSONP4、JSONP5、JSONP6、JSONP7、JSONP8、JSONP9、JSONP10、JSONP11、JSONP12、JSONP13、JSONP14、JSONP15、JSONP16、JSONP17、JSONP18、JSONP19、JSONP20、JSONP21、JSONP22、JSONP23、JSONP24、JSONP25、JSONP26、JSONP27、JSONP28、JSONP29、JSONP30、JSONP31、JSONP32、JSONP33、JSONP34、JSONP35、JSONP36、JSONP37、JSONP38、JSONP39、JSONP40、JSONP41、JSONP42、JSONP43、JSONP44、JSONP45、JSONP46、JSONP47、JSONP48、JSONP49、JSONP50、JSONP51、JSONP52、JSONP53、JSONP54、JSONP55、JSONP56、JSONP57、JSONP58、JSONP59、JSONP60、JSONP61、JSONP62、JSONP63、JSONP64、JSONP65、JSONP66、JSONP67、JSONP68、JSONP69、JSONP70、JSONP71、JSONP72、JSONP73、JSONP74、JSONP75、JSONP76、JSONP77、JSONP78、JSONP79、JSONP80、JSONP81、JSONP82、JSONP83、JSONP84、JSONP85、JSONP86、JSONP87、JSONP88、JSONP89、JSONP90、JSONP91、JSONP92、JSONP93、JSONP94、JSONP95、JSONP96、JSONP97、JSONP98、JSONP99、JSONP100、JSONP101、JSONP102、JSONP103、JSONP104、JSONP105、JSONP106、JSONP107、JSONP108、JSONP109、JSONP110、JSONP111、JSONP112、JSONP113、JSONP114、JSONP115、JSONP116、JSONP117、JSONP118、JSONP119、JSONP120、JSONP121、JSONP122、JSONP123、JSONP124、JSONP125、JSONP126、JSONP127、JSONP128、JSONP129、JSONP130、JSONP131、JSONP132、JSONP133、JSONP134、JSONP135、JSONP136、JSONP137、JSONP138、JSONP139、JSONP140、JSONP141、JSONP142、JSONP143、JSONP144、JSONP145、JSONP146、JSONP147、JSONP148、JSONP149、JSONP150、JSONP151、JSONP152、JSONP153、JSONP154、JSONP155、JSONP156、JSONP157、JSONP158、JSONP159、JSONP160、JSONP161、JSONP162、JSONP163、JSONP164、JSONP165、JSONP166、JSONP167、JSONP168、JSONP169、JSONP170、JSONP171、JSONP172、JSONP173、JSONP174、JSONP175、JSONP176、JSONP177、JSONP178、JSONP179、JSONP180、JSONP181、JSONP182、JSONP183、JSONP184、JSONP185、JSONP186、JSONP187、JSONP188、JSONP189、JSONP190、JSONP191、JSONP192、JSONP193、JSONP194、JSONP195、JSONP196、JSONP197、JSONP198、JSONP199、JSONP200、JSONP201、JSONP202、JSONP203、JSONP204、JSONP205、JSONP206、JSONP207、JSONP208、JSONP209、JSONP210、JSONP211、JSONP212、JSONP213、JSONP214、JSONP215、JSONP216、JSONP217、JSONP218、JSONP219、JSONP220、JSONP221、JSONP222、JSONP223、JSONP224、JSONP225、JSONP226、JSONP227、JSONP228、JSONP229、JSONP230、JSONP231、JSONP232、JSONP233、JSONP234、JSONP235、JSONP236、JSONP237、JSONP238、JSONP239、JSONP240、JSONP241、JSONP242、JSONP243、JSONP244、JSONP245、JSONP246、JSONP247、JSONP248、JSONP249、JSONP250、JSONP251、JSONP252、JSONP253、JSONP254、JSONP255、JSONP256、JSONP257、JSONP258、JSONP259、JSONP260、JSONP261、JSONP262、JSONP263、JSONP264、JSONP265、JSONP266、JSONP267、JSONP268、JSONP269、JSONP270、JSONP271、JSONP272、JSONP273、JSONP274、JSONP275、JSONP276、JSONP277、JSONP278、JSONP279、JSONP280、JSONP281、JSONP282、JSONP283、JSONP284、JSONP285、JSONP286、JSONP287、JSONP288、JSONP289、JSONP290、JSONP291、JSONP292、JSONP293、JSONP294、JSONP295、JSONP296、JSONP297、JSONP298、JSONP299、JSONP300、JSONP301、JSONP302、JSONP303、JSONP304、JSONP305、JSONP306、JSONP307、JSONP308、JSONP309、JSONP310、JSONP311、JSONP312、JSONP313、JSONP314、JSONP315、JSONP316、JSONP317、JSONP318、JSONP319、JSONP320、JSONP321、JSONP322、JSONP323、JSONP324、JSONP325、JSONP326、JSONP327、JSONP328、JSONP329、JSONP330、JSONP331、JSONP332、JSONP333、JSONP334、JSONP335、JSONP336、JSONP337、JSONP338、JSONP339、JSONP340、JSONP341、JSONP342、JSONP343、JSONP344、JSONP345、JSONP346、JSONP347、JSONP348、JSONP349、JSONP350、JSONP351、JSONP352、JSONP353、JSONP354、JSONP355、JSONP356、JSONP357、JSONP358、JSONP359、JSONP360、JSONP361、JSONP362、JSONP363、JSONP364、JSONP365、JSONP366、JSONP367、JSONP368、JSONP369、JSONP370、JSONP371、JSONP372、JSONP373、JSONP374、JSONP375、JSONP376、JSONP377、JSONP378、JSONP379、JSONP380、JSONP381、JSONP382、JSONP383、JSONP384、JSONP385、JSONP386、JSONP387、JSONP388、JSONP389、JSONP390、JSONP391、JSONP392、JSONP393、JSONP394、JSONP395、JSONP396、JSONP397、JSONP398、JSONP399、JSONP400、JSONP401、JSONP402、JSONP403、JSONP404、JSONP405、JSONP406、JSONP407、JSONP408、JSONP409、JSONP410、JSONP411、JSONP412、JSONP413、JSONP414、JSONP415、JSONP416、JSONP417、JSONP418、JSONP419、JSONP420、JSONP421、JSONP422、JSONP423、JSONP424、JSONP425、JSONP426、JSONP427、JSONP428、JSONP429、JSONP430、JSONP431、JSONP432、JSONP433、JSONP434、JSONP435、JSONP436、JSONP437、JSONP438、JSONP439、JSONP440、JSONP441、JSONP442、JSONP443、JSONP444、JSONP445、JSONP446、JSONP447、JSONP448、JSONP449、JSONP450、JSONP451、JSONP452、JSONP453、JSONP454、JSONP455、JSONP456、JSONP457、JSONP458、JSONP459、JSONP460、JSONP461、JSONP462、JSONP463、JSONP464、JSONP465、JSONP466、JSONP467、JSONP468、JSONP469、JSONP470、JSONP471、JSONP472、JSONP473、JSONP474、JSONP475、JSONP476、JSONP477、JSONP478、JSONP479、JSONP480、JSONP481、JSONP482、JSONP483、JSONP484、JSONP485、JSONP486、JSONP487、JSONP488、JSONP489、JSONP490、JSONP491、JSONP492、JSONP493、JSONP494、JSONP495、JSONP496、JSONP497、JSONP498、JSONP499、JSONP500、JSONP501、JSONP502、JSONP503、JSONP504、JSONP505、JSONP506、JSONP507、JSONP508、JSONP509、JSONP510、JSONP511、JSONP512、JSONP513、JSONP514、JSONP515、JSONP516、JSONP517、JSONP518、JSONP519、JSONP520、JSONP521、JSONP522、JSONP523、JSONP524、JSONP525、JSONP526、JSONP527、JSONP528、JSONP529、JSONP530、JSONP531、JSONP532、JSONP533、JSONP534、JSONP535、JSONP536、JSONP537、JSONP538、JSONP539、JSONP540、JSONP541、JSONP542、JSONP543、JSONP544、JSONP545、JSONP546、JSONP547、JSONP548、JSONP549、JSONP550、JSONP551、JSONP552、JSONP553、JSONP554、JSONP555、JSONP556、JSONP557、JSONP558、JSONP559、JSONP560、JSONP561、JSONP562、JSONP563、JSONP564、JSONP565、JSONP566、JSONP567、JSONP568、JSONP569、JSONP570、JSONP571、JSONP572、JSONP573、JSONP574、JSONP575、JSONP576、JSONP577、JSONP578、JSONP579、JSONP580、JSONP581、JSONP582、JSONP583、JSONP584、JSONP585、JSONP586、JSONP587、JSONP588、JSONP589、JSONP590、JSONP591、JSONP592、JSONP593、JSONP594、JSONP595、JSONP596、JSONP597、JSONP598、JSONP599、JSONP600、JSONP601、JSONP602、JSONP603、JSONP604、JSONP605、JSONP606、JSONP607、JSONP608、JSONP609、JSONP610、JSONP611、JSONP612、JSONP613、JSONP614、JSONP615、JSONP616、JSONP617、JSONP618、JSONP619、JSONP620、JSONP621、JSONP622、JSONP623、JSONP624、JSONP625、JSONP626、JSONP627、JSONP628、JSONP629、JSONP630、JSONP631、JSONP632、JSONP633、JSONP634、JSONP635、JSONP636、JSONP637、JSONP638、JSONP639、JSONP640、JSONP641、JSONP642、JSONP643、JSONP644、JSONP645、JSONP646、JSONP647、JSONP648、JSONP649、JSONP650、JSONP651、JSONP652、JSONP653、JSONP654、JSONP655、JSONP656、JSONP657、JSONP658、JSONP659、JSONP660、JSONP661、JSONP662、JSONP663、JSONP664、JSONP665、JSONP666、JSONP667、JSONP668、JSONP669、JSONP670、JSONP671、JSONP672、JSONP673、JSONP674、JSONP675、JSONP676、JSONP677、JSONP678、JSONP679、JSONP680、JSONP681、JSONP682、JSONP683、JSONP684、JSONP685、JSONP686、JSONP687、JSONP688、JSONP689、JSONP690

译者序

非常荣幸可以参与翻译《jQuery in Action》第三版，这是jQuery之父John Resig和jQuery基金会主席Dave Methvin强烈推荐的书。此书是jQuery领域最经典的学习书籍，没有之一，也是曼宁（Manning）出版社的“××× in Action”系列的经典书籍之一。本书由jQuery领域的技术专家和开发团队成员联合编写，受到全球读者的五星好评。我翻译的另外一本《MongoDB in Action》第二版，同样是一本经典书籍，也即将出版。

jQuery作为最流行的JavaScript框架，其理念为Write less, do more（写得少，做得多），提倡编写少量的代码实现复杂的功能；它简单、快速、轻量级，能提供丰富的功能接口。对于前端开发最头痛的浏览器兼容性问题，jQuery提供了很好的底层封装，以适应不同浏览器版本的差异；对于Ajax、动画、DOM操作、事件、Promise、闭包、扩展插件、测试等提供了丰富的支持。简单易用，减轻了程序员底层兼容性开发的痛苦，适用于现代Web网站快速开发的需求。学习Web网站开发，jQuery技术成为必备技能。全球排名前1000万的网站有63%使用了jQuery，其重要性不言而喻。

《jQuery实战》适合于想深入学习jQuery的开发人员。本书名字虽然包含“实战”，但是并非基础书籍，它深入浅出，通过大量的实例代码来介绍jQuery框架底层的实现，比如事件原理、CSS选择器、动画机制、jQuery扩展自定义插件编程，以及其他的开发工具和框架，如Bower和QUnit，当然还有大型Web项目经典开发原则。

我学习技术开发的原则就是坚持阅读国外经典书籍，其次就是阅读官方文档。这个习惯让我受益匪浅。本书就是我们学习jQuery高级开发的经典书籍。

移动互联网时代，全栈工程师（Full Stack Engineer）很受欢迎。但是他们在学习技术时片面追求广度，忽略深度，因此不利于职业的长期发展。编程语言并无优劣之分，总是有许多共性或相似点，在学习过程中，应尽量去融会贯通，吸取其精华。作为高级开发人员，应该具备或理解框架底层的工作原理，以便于实际工作中的分析和优化代码。

在业余时间翻译本书的过程中，我还编写代码、作为架构顾问参与设计架构方案。翻译本书的过程中，我收获很多，重新认识了jQuery框架，包括底层的实现细节。

感谢徐扬帮助翻译了部分章节，也要特别感谢华中科技大学出版社编辑们的辛苦工作，是他们才使得本书的翻译工作可以顺利完成，能按期与读者见面。

本书与《MongoDB实战》将作为新青年X软件训练营（54peixun.com）高级架构班的推荐教材，希望本书能帮助大家深入学习jQuery框架。欢迎加入读者QQ群149395911，或者微博@徐雷FrankXuLei联系我。

徐雷（Frank Xu Lei）

2016年7月

译者介绍

- (1) 新青年X软件训练营（54peixun.com）联合创始人。
- (2) 微软中国MSDN特邀讲师、微软美国Channel9首位中国讲师。
- (3) 微软大企业客户技术顾问，曾任购酒网、元拓商城、上海市公安局ERP系统技术架构顾问。
- (4) 获得吉林大学计算机科学与技术学士学位，上海交通大学硕士学位。
- (5) 国外经典《WCF技术内幕》、《WCF服务编程》（第三版、第四版）、《ASP.NET MVC4 Web编程》、《jQuery实战》（第三版）、《MongoDB实战》（第二版）的译者。
- (6) 受邀在微软中国、盛大网络、玫琳凯中国研发中心、世界500强约翰迪尔、一嗨租车、沪江网、中国东方航空、美国IGT、世界500强花旗银行、达丰集团、上海交通大学软件学院、中国体彩集团总部等中外名企、名校授课。
- (7) 2014年9月16日，受邀作为微软中国开发者技术代表会见ASP.NET之父Scott Gu先生！
- (8) 苍老师忠实粉丝，授课幽默风趣，追求“德艺双馨”。

第三版推荐序

Foreword to the third edition

10年前, John Resig发明了一种可以简化Web开发的JavaScript框架。今天, 根据BuiltWith.com的统计, 超过80%使用JavaScript的网站使用了jQuery。如果你不知道jQuery, 就很难称得上是一个Web开发人员。

从技术角度来说, jQuery大大简化了浏览器原生调用的方法代码, 压缩了功能性代码量。这就是jQuery的口号“Write less, do more”(写得少, 做得多)。jQuery框架也屏蔽了一些兼容性行为问题, 以及一些浏览器中存在的著名的bug, 大大简化了开发和测试工作。

设计之初, jQuery就非常易于扩展。jQuery的插件模型让每个开发者都可以在jQuery之上构建自己的扩展功能, 有成千上万从广告效果到表单验证的jQuery插件。这样, 许多Web开发者都可以很容易基于大量的开源插件构建完美的网站。

仅靠代码并不能让jQuery如此流行。强大的在线论坛和邮件组支持开发者在线回答问题是jQuery流行的重要原因。这些深入讨论交流都促进了文档、培训课程和编程书籍的发展。

本书是学习jQuery开发的绝佳途径。前面章节介绍了jQuery框架网页开发需要的核心API, 这些API可以选择页面元素进行操作。同样的模式适合隐藏、展示、动画、删除或修改元素的外观操作。选择元素的过程中使用了选择器语法, 而jQuery框架提供了很强大的封装支持。

必须承认, “事件”这一章是我最喜欢的, 因为我参与了jQuery 1.7中event模块的核心代码开发工作。这一章非常精彩, 详细解释了事件的目的及其在网页中的用途。通过本章可以了解用户与网页交互的原理。几乎任何一项jQuery操作都是从某种特定的事件开始的。

非常高兴, 本书还增加了一些容易被忽视的内容, 比如单元测试与大型项目的结构。许多小项目最终都会变成大项目, 本书的内容也可以帮助读者学习如何管理持续不断增加的项目。

本书编写了很多demo程序, 向读者展示了jQuery各个模块如何协同工作, 也演示了模板的概念、所有最新JavaScript框架和应用程序的核心。即使是现在, 我也感觉非常惊奇, 居然可以使用如此少的代码实现各种不同功能的强大demo!

Aurelio De Rosa已经为jQuery社区贡献了很多年, 也是jQuery内容团队成员, 致力于确保jQuery在线文档的及时准确更新。他在本书中给大家最新的jQuery库的信息。Aurelio在编写本书的时候也完善了在线文档, 以及一些不一致或者缺失的内容。作为本书的读者, 你真的非常幸运, 很快就会成为真正的jQuery开发者。加油! “Write less, do more!”

Dave Methvin
jQuery基金会主席

第一版推荐序

Foreword to the first edition

大道至简。为什么创建简单的页面交互效果非要编写臃肿、复杂的代码呢？复杂性并不是开发Web应用的必要条件。

当初我创建jQuery项目时，决定简化Web开发者日常的编码工作。当阅读《jQuery实战》这本书时，我非常高兴地看到这本书深入、准确地介绍了jQuery框架的精髓。

对于像学习实际简化代码编写工作的人来说，《jQuery实战》确实是梦寐以求的学习资料。

最让我开心的是，本书中Bear和Yehuda非常详细地介绍了jQuery框架的内部机制、原理。他们深入研究了jQuery API。他们每天都会通过邮件或者即时聊天工具询问关于框架的问题，包括反馈他们发现的各种bug。当收不到他们的消息时，我反而感觉非常失落。可以放心阅读，本书应该是你看过的jQuery领域最好的学习书籍，完整介绍和深入研究了jQuery框架。

最让我惊喜的是，本书是关于jQuery扩展插件开发主体的，详细介绍了jQuery插件的开发步骤和原理。为什么jQuery如此简单，就是因为使用了插件架构，提供了许多可以扩展插件的功能点。通常来说，有些功能非常有用，但不是普遍使用，没有办法直接包含在jQuery类库里，最后的方式就是使用插件架构。本书中介绍了一些插件，比如Forms、Dimension及Live-Query，这些插件已经被广泛使用，原因很简单：它们开发专业，文档详细且持续维护。记得一定要特别注意插件的使用方法及开发原理，这是jQuery开发的基础。

有《jQuery实战》这样的经典书籍，相信jQuery项目会日新月异，更加辉煌。希望本书可以帮助你深入学习jQuery的开发知识和底层原理，让你如虎添翼。

John Resig

jQuery之父

序言

Preface

每次回想起为本书付出的巨大心血，我都会吓一跳。曼宁出版社的人联系我，我希望我参与《jQuery in Action》（第三版）的编写工作时，我虽然知道这并不是一件轻松的差事，但是，我还是低估了这个工作量。我原以为：“这毕竟是小菜一碟。几个月就能搞定了。”其实是两年时间，夜以继日地工作才完成。但是我没有后悔当初的选择。编写本书已经成为难以忘怀的经历，它改变了我，锻炼了我多方面的能力。我已经成为一位更出色的工程师和作者，当然也使我的jQuery编程水平得以精进。

两年前，我还是一个钟情于jQuery的开发者，非常感谢这个框架解决了我许多棘手的技术问题。编写本书之前，我的jQuery知识已经非常不错，但是参与本书的编写让我对jQuery知识了解得更加深入，学习了很多底层的架构，更上了一层楼。修订本书要求我定期为jQuery项目作出贡献——我已经被邀请加入jQuery团队。毋庸置疑，参与本书的编写工作，是我人生中意想不到的荣誉和巨大成就，我非常自豪。

既然已经知道我为什么参与本书的编写，那么还有一个关键的问题：有没有必要编写第三版？我认为是有必要的。主要有两个基本原因：首先，第二版涵盖的最新内容为jQuery 1.4框架，第三版覆盖最新的1.11版内容，jQuery 3(包含在本书中)刚刚开始。其次，jQuery是最流行的JavaScript框架，世界上排名前100万的网站中有63%的使用了jQuery，从第二版出版后jQuery的变化很大。这两个原因要求我们必须编写更新的书籍，jQuery并非停滞不前，更不会无故消失。

xxi

第三版中你会看到一些变化。首先，删除了关于jQuery UI的章节，因为jQuery和jQuery UI不断发展壮大，应该有各自的著述。其次，正如你翻开本书看到的一样，我决定增加一些第二版没有包含的高级主题。最后，还介绍了一些例子，如试验页面、代码片段、在线demo及其他内容。翻过这一页，深入学习本书，开始学习这个世界上最流行的JavaScript框架吧。祝你学习愉快！

我把本书特别献给你，我亲爱的Annunziata。谢谢在一起度过的所有美好的时刻和那些即将到来的甜蜜生活。我爱你，宝贝。

Aurelio De Rosa xxii

我还要特别感谢我的家人，Raffaella, Eufemia, Camilla, Nicola。我的奶奶 Rosalinda和Anna，以及我的爷爷Aurelio。你们无微不至的关心和支持，让我能安心工作。谢谢你们太多。

我还想感谢Francesco Palladino。你是我最好的朋友，从那时起，你让我的生活变得美好、美妙。

致谢

Acknowledgments

正如前一个版本一样，每一个参与出版工作的人都作出了不可磨灭的贡献。写一本好书不只需要作者需要花费很多的时间，编辑、设计等其他工作人员同样也付出了不少心血。

曼宁出版社的工作人员不知疲倦地工作，以确保本书达到预期的质量水平，我要感谢他们付出的努力。没有他们的努力，本书不会及时出版。要感谢的不仅仅是出版社的Marjan Bace，还包括Al Scherer、Ana Romac、Candace Gillhoolley、Cynthia Kane、Dottie Marsico、Jeff Bleiel、Kevin Sullivan、Linda Recktenwald、Mary Piergies、Melody Dolab、Ozren Harlovic、Robin de Jongh、Scott Meyers及Sean Dennis。非常感谢他们及许多幕后工作人员的辛勤劳动。

其次要感谢那些参与本书审阅的朋友，他们纠正了很多拼写和技术代码方面的错误。他们的贡献同样巨大。非常感谢Chris Maki、Christopher Haupt、Chuck Durfee、Francesco Bianchi、Gary A. Stafford、Gregor Zurowski、Jan Goyvaerts、Jean-Francois Morin、John D. Lewis、John Stemper、Karen Christenson、Keith Webster、Matt Forsythe、Ricardo Mano、Ryan Meeks、Suraj Kumar、William E. Wheeler及Willie Roberts。

特别感谢本书的技术校订员Richard Scott-Robinson，他花费了大量时间来检查本书的每个例子代码，以确保代码能够在不同的环境中运行。他同样为本书的内容提出了很多深入准确的个人见解。

xxiii

非常感谢jQuery基金会主席Dave Methvin为本书所撰写的推荐序，并且支持认同我的工作，前两位作者Bear Bibeault和Yehuda Katz在第二版畅销后继续努力，必将使第三版再创辉煌。

个人方面，我最想感谢的是我的未婚妻Annarita小姐。你甜蜜的爱和耐心陪我度过了本书编写的时光，此爱永久。在我忙于本书编写的两年时间里，你从来没有抱怨过一次。善解人意的姑娘，我想把本书特别献给你，我亲爱的Annarita。谢谢在一起度过的所有美好的时刻和那些即将到来的甜蜜生活。我爱你，宝贝。

还要特别感谢我的家人：Raffaele、Eufemia、Giusy、Viola、我的奶奶Giuseppina和Anna，以及我的爷爷Aurelio。你们无微不至的关心和支持，让我能安心工作，亏欠你们太多。

我还想感谢Francesco Palladino，你是我最好的朋友，总能随叫随到。我祝福你生活美满，美梦成真。

说到梦想，我想把本书献给那些雄心勃勃、追逐梦想的人。不要因为别人的嘲笑妄自菲薄，尽管困难重重。有一天你终究会成功。本书献给所有追逐梦想的人，祝福你们。

还要感谢那些培养和教育过我的人，是你们帮助我不断成长：Albert Einstein、Ludwig van Beethoven、Lucius Annaeus Seneca、Roberto De Rosa、Leonardo Grisolia及那个无名的卖伞人。

最后，我要感谢jQuery团队的所有成员。如果我编写了一本好书，那是因为你们这些年的辛勤付出，你们很棒！

xxiv

Aurelio De Rosa

关于本书

about this book

本书适合想深入学习jQuery的Web开发人员。jQuery是互联网上最流行的JavaScript框架。本书的目标是希望读者成为Web高级开发人员，无论起点如何。本书深入介绍了整个jQuery框架，此外还专门深入介绍了插件编程，以及一些扩展开发工具和框架，比如Bower和QUnit，当然还有经典的开发实战原则。每个API方法都使用了简明扼要的语法块来描述参数和返回值。

《jQuery实战》（第三版）涵盖了从简单入门（如何在网页中引入jQuery）到高级开发的内容，比如Promises的实现方式，以及如何开发jQuery插件。为了便于大家理解知识，本书包含了大量的实例代码、三个插件及三个例子项目。本书也包含了试验网页（Lab Pages）。这些有趣的网页开发可以让大家在实战开发中快速掌握jQuery方法的差别，而不需要编写大量的代码。

阅读本书需要大家提前掌握HTML、CSS和JavaScript的基础编程知识。jQuery以前的知识不是必需的，但是可以帮助大家快速理解掌握新的概念。

路线图

本书分为三个部分：jQuery入门、jQuery核心（包含了所有的特性）及高级主题。

xxv

第1章主要介绍了jQuery框架背后的原理以及原则；讨论了jQuery框架的本质，以及它要解决的问题，为什么要在Web项目中使用jQuery框架。

第2章讲述了使用选择器查找DOM元素，以及如何创建自定义选择器；也介绍了jQuery集合（以及jQuery对象）等词汇，即jQuery方法返回的对象。它包含使用jQuery库操作的元素。

第3章扩展了第2章的内容，涉及如何通过前一个元素创建新的选择元素，也介绍了如何通过jQuery创建新的选择。

第4章关注如何使用jQuery提供的操作特性和属性的方法，以及这些方法的差别，此外，还解释了如何在一个或者多个DOM元素上存储自定义数据。

第5章介绍了如何使用class名字操作元素，如何克隆、设置DOM元素，如何通过添加、移动和替换来修改DOM树。

第6章介绍了各种不同的事件模型，以及浏览器如何建立事件处理器来响应事件发生的处理工作，之后介绍了jQuery如何支持这种机制，避免开发人员陷入浏览器兼容性的大坑中。此外，本章还介绍了两个重要的主题：事件委托（event delegation）和事件冒泡（event bubbling）。

第7章与前面几章的内容不同，本章的目标主要是带领大家开发一个Web项目：DVD光盘定位器（DVD discs locator），可以在这里把所学的知识应用其中，并实战练习。

第8章介绍了显示和隐藏元素的方法，以及如何创建动画效果，实现连续运行效果的函数队列，以及一些常见的函数。

第9章主要介绍了工具函数，这些函数使用了jQuery命名空间，但不直接操作DOM元素。

第10章的内容涵盖了最近几年最重要的概念Ajax。我们来学习jQuery如何简化Ajax编程。为了避免程序员掉入坑中，jQuery简化了最常见的Ajax交互类型（比如返回JSON对象）。

第11章设置了新的挑战。要解决许多开发者面对的真实问题：创建一个联系表单。项目包括创建一个可以工作的表单页面，而不需要通过完全刷新加载页面来告诉用户是否成功或者失败。

第12章是第三部分的第一章，从本章开始我们进入了高级主题，大部分的内容并非与核心库相关。本章讨论了如何通过创建插件来扩展jQuery功能。插件主要分为两大类：方法和工具函数。本章详细讲解了这些内容。

第13章介绍了如何通过jQuery Promises避免臭名昭著的回调灾难（callback hell）问题。当然，众所周知，这是一个略有争议的话题，已经持续了很多年。

第14章介绍了测试：什么是测试及测试为什么如此重要。首先会关注一种特定类型的测试：单元测试（unit testing）。然后会介绍非常流行的框架QUnit，它使用了一些jQuery项目（jQuery、jQuery UI和jQuery Mobile）来测试代码。

第15章是本书的最后一章，先介绍了jQuery开发与性能优化的技巧，然后扩展到几个与jQuery相关的工具、框架和模式，它们可以帮助我们创建快速、强壮和优美的JavaScript代码。特别要强调的是，本章会讲解如何在模块中组织代码、如何使用RequireJS加载代码，以及如何使用Bower来管理前端依赖。最后会通过Backbone.js向大家揭秘jQuery如何开发单页面Web应用程序。

最后，本书末尾提供了专门的附录来列举JavaScript的概念，比如函数上下文和闭包——如何最高效地使用jQuery开发网页，为不熟悉或者想重新温习这些概念的读者服务。

代码规范和下载

本书使用固定宽度的字体作为参考代码的专用字体。这些列举的代码主要是为了方便理解关

键的知识点，有些内容使用编号列也是为了对代码进行补充说明，通过换行或者缩进也是为了让排版充分使用页面空间。

本书中的所有代码都可以在GitHub上下载，地址是：<https://github.com/AurelioDeRosa/jquery-in-action>。当然也可以从出版社的网站www.manning.com/derosa/或www.manning.com/jquery-in-action-third-edition下载。

软件需求

本书的例子代码都在每章的一个文件夹中，可以方便地在Apache HTTP Server中托管运行。除了第7章、第10章的代码及其他章节的少量代码，其他代码都可以直接在浏览器中运行调试。第10章的代码运行配置环境有点复杂，不仅需要与Apache后台交互，还需要配置PHP环境（如果使用的是其他开发框架，比如Node.js、JSP或者ASP.NET，一样可以使用jQuery框架，不限制后台框架，完全兼容）。

所有的例子都已在各种不同的浏览器中测试完成，包括IE、Firefox、Safari、Opera和Chrome。 xxvii

关于封面插图

《jQuery实战》（第三版）的封面图片标题是“The Watchman”（警卫）。

此图片选自于法国旅行家G. St. Saveur 200年前编写出版的旅行书籍：《旅游百科全书》（*Encyclopédie des Voyages*）。旅游在当时还是一种新兴的社会现象，这种旅行指南非常受欢迎，不仅介绍了旅行者，还介绍了世界不同地区的居民和法国士兵、公务员、商人以及农民。

《旅游百科全书》中的插图非常生动地再现了200年前世界各地不同的风貌。

由于彼此隔绝，不同地区的人说着不同的语言。在大街上或者乡下，很容易辨别他们的家乡，也可以通过服装判断他们的职业。

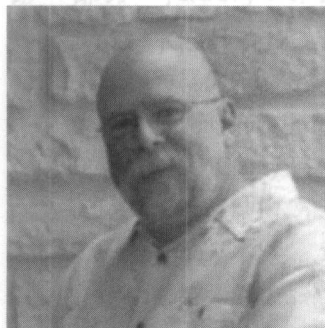
从那以后衣着也发生了很大变化，那个丰富多彩的时代已经远离我们了。现在很难鉴别不同地区和国家的人民。也许我们已经牺牲了文化的多样性来换取个人生活的多样性——更多样化、更快节奏的现代生活。

现在已经很难区分不同的计算机书籍，20年前，曼宁出版社开始采用多样性的区域生活图画来作为图书封面的设计创意，通过图书封面来介绍世界各地不同的历史文化与人文生活，让我们感受不同历史时期的文化，正如本书一样。

xxviii

作者简介

about the authors

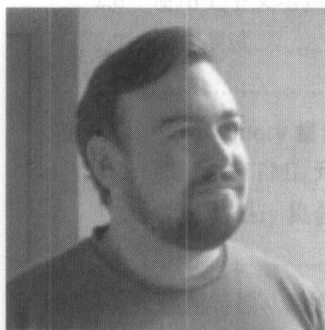


Bear Bibeault从最早的100波特率的Control Data Cyber计算机编写井字游戏开始，编写了超过30年的代码。因为他获得了两个电气工程学位，所以大家认为Bear应该设计天线或者其他电器设备。但是从第一份在美国数据设备公司（Digital Equipment Corporation）的工作开始，他就开始迷恋上了编程工作。

Bear曾经就职于Lightbridge、BMC Software、Dragon Systems、Works.com等公司。Bear也曾经就职于美国陆军部队，训练士兵如何炸毁坦克。现在他作为高级Web开发工程师，就职于一家领先的对象存储软件公司。

工作之余，Bear也会撰写书稿，经营一家小公司，开发网站和媒体服务（不是婚礼摄像），作为高级版主也帮助管理JavaRanch.com网站。Bear喜欢做大餐，痴迷摄影和录像，骑他的Yamaha V-Star摩托车兜风，穿着印有热带雨林图案的T恤。

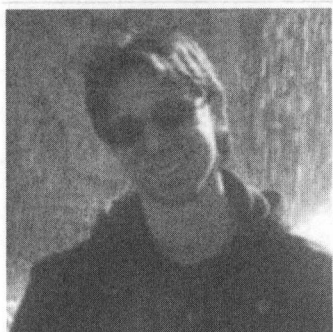
他现在工作在得克萨斯州的奥斯汀（Austin）市，他非常喜欢Austin，除了那些疯狂的司机。 < xxix



Yehuda Katz在过去几年参与过许多开源项目。除了作为jQuery项目的核心成员，他还参与了Merb项目——Ruby on Rails的一种替代框架（也是使用Ruby开发）。

Yehuda出生于明尼苏达州，在纽约长大，现居住在加州的圣巴巴拉市。他为《New York Times》、《Allure Magazine》、《Architectural Digest》、《Yoga Journal》等高端客服提供解决方案。他擅长Java、Ruby、PHP和JavaScript开发语言和框架。

在空闲时间，他还维护VisualjQuery.com网站，帮助回答IRC频道里jQuery用户和jQuery邮件列表的技术问题。



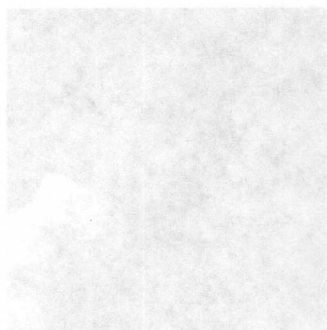
Aurelio De Rosa是一名全栈Web开发工程师，拥有WAMP stack与HTML5、CSS3、Sass、JavaScript和PHP超过5年的专业开发经验。他是jQuery和JoindIn团队的成员、JavaScript和HTML5领域的专家。他同样对Web安全、访问性、性能和SEO感兴趣。

不忙的时候，他也撰写技术文章、演讲、写书，还是一些学术论文的共同作者。

XXX

从最早的Control Data Cybernetics到现在的Biscuits，Aurelio在计算机行业工作已经超过30年。他最初是作为一名硬件工程师，后来转行成为一名软件工程师。他在Biscuits公司工作期间，负责设计和开发各种嵌入式系统。他还在Biscuits公司担任过项目经理，负责协调团队完成各种项目。他现在是Biscuits公司的首席工程师，负责公司的技术战略和开发工作。

Aurelio在Biscuits公司工作期间，还负责过Biscuits公司的软件开发工作。他负责过Biscuits公司的各种嵌入式系统，包括各种工业控制设备、医疗设备、以及各种消费电子产品。他还在Biscuits公司担任过项目经理，负责协调团队完成各种项目。他现在是Biscuits公司的首席工程师，负责公司的技术战略和开发工作。



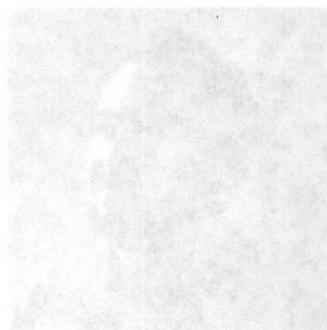
一家名为Biscuits的公司。

Aurelio在Biscuits公司工作期间，还负责过Biscuits公司的软件开发工作。他负责过Biscuits公司的各种嵌入式系统，包括各种工业控制设备、医疗设备、以及各种消费电子产品。他还在Biscuits公司担任过项目经理，负责协调团队完成各种项目。他现在是Biscuits公司的首席工程师，负责公司的技术战略和开发工作。

XXX

Aurelio在Biscuits公司工作期间，还负责过Biscuits公司的软件开发工作。他负责过Biscuits公司的各种嵌入式系统，包括各种工业控制设备、医疗设备、以及各种消费电子产品。他还在Biscuits公司担任过项目经理，负责协调团队完成各种项目。他现在是Biscuits公司的首席工程师，负责公司的技术战略和开发工作。

Aurelio在Biscuits公司工作期间，还负责过Biscuits公司的软件开发工作。他负责过Biscuits公司的各种嵌入式系统，包括各种工业控制设备、医疗设备、以及各种消费电子产品。他还在Biscuits公司担任过项目经理，负责协调团队完成各种项目。他现在是Biscuits公司的首席工程师，负责公司的技术战略和开发工作。



Aurelio在Biscuits公司工作期间，还负责过Biscuits公司的软件开发工作。他负责过Biscuits公司的各种嵌入式系统，包括各种工业控制设备、医疗设备、以及各种消费电子产品。他还在Biscuits公司担任过项目经理，负责协调团队完成各种项目。他现在是Biscuits公司的首席工程师，负责公司的技术战略和开发工作。

目录

Table of Contents

第一部分 jQuery 入门	1
第 1 章 jQuery 介绍	3
1.1 写得少, 做得多	4
1.2 低调的 JavaScript	5
1.3 安装 jQuery	7
1.4 jQuery 结构	11
1.5 jQuery 本质	13
1.6 总结	17
第二部分 jQuery 核心	19
第 2 章 选择元素	21
2.1 选择操作元素	21
2.2 基本选择器	23
2.3 根据层级查找元素	28
2.4 通过属性来选择元素	30
2.5 过滤器介绍	33
2.6 使用上下文增强性能	43
2.7 技能测试	45
2.8 总结	46
第 3 章 操作 jQuery 集合	47
3.1 创建新 HTML 元素	47
3.2 管理 jQuery 集合	50
3.3 总结	70
第 4 章 使用特性、属性和数据	71
4.1 定义元素特性和属性	71
4.2 使用属性	74
4.3 操作元素特性	79

4.4	元素中存储自定义数据	82
4.5	总结	89
第 5 章	使用 jQuery 操作页面	90
5.1	修改元素的样式	90
5.2	设置元素内容	103
5.3	处理表单元素的值	119
5.4	总结	121
第 6 章	事件本质	122
6.1	理解浏览器事件模型	123
6.2	jQuery 事件模型	135
6.3	总结	154
第 7 章	DVD 光盘定位器	156
7.1	让事件开始工作	156
7.2	总结	170
第 8 章	使用动画与特效	171
8.1	显示和隐藏元素	172
8.2	动画元素的显示状态	175
8.3	为 jQuery 添加更多 easing 函数	185
8.4	创建自定义动画	188
8.5	动画与排队	192
8.6	总结	201
第 9 章	jQuery 工具函数操作 DOM	203
9.1	使用 jQuery 属性	204
9.2	通过 jQuery 使用其他库	207
9.3	操作 JavaScript 对象和集合	210
9.4	其他工具函数	229
9.5	总结	234
第 10 章	使用 Ajax 与服务器交互	236
10.1	复习 Ajax	236
10.2	加载内容到元素中	241
10.3	发送 GET 和 POST 请求	249
10.4	完全控制 Ajax 请求	261
10.5	总结	270

第 11 章	demo:Ajax 驱动的联系方式表单	272
11.1	项目功能	272
11.2	创建页面标签	274
11.3	实现 PHP 后台	276
11.4	使用 Ajax 验证字段	277
11.5	Ajax 更多乐趣	279
11.6	使用动画特效改善用户体验	281
11.7	注意访问性	282
11.8	总结	283
第三部分	高级主题	285
第 12 章	jQuery 扩展插件	287
12.1	为什么扩展 jQuery	287
12.2	在哪里查找插件	288
12.3	jQuery 插件编写指南	292
12.4	演示: 创建 jQuery 幻灯片插件	306
12.5	编写自定义工具函数	316
12.6	总结	321
第 13 章	使用 Deferred 避免回调地狱	322
13.1	promise 介绍	322
13.2	Deferred 与 Promise 对象	326
13.3	Deferred 方法	326
13.4	promise 化一切	344
13.5	总结	345
第 14 章	使用 QUnit 进行单元测试	347
14.1	为什么测试很重要	347
14.2	QUnit 入门	350
14.3	创建同步测试	353
14.4	使用断言测试代码	355
14.5	如何测试异步任务	361
14.6	noglobals 与 notrycatch	363
14.7	模块分组测试	365
14.8	配置 QUnit	366
14.9	测试套件的例子	367

14.10	总结	371
第 15 章	jQuery 大型项目开发	373
15.1	改进选择器性能	374
15.2	使用模块组织代码	378
15.3	使用 RequireJS 加载模块	381
15.4	使用 Bower 管理依赖	385
15.5	使用 Backbone.js 创建单页应用	389
15.6	总结	403
15.7	结尾	404
附录 A	JavaScript 高级编程必备知识	405
A.1	JavaScript 对象基础	405
A.2	一等公民函数	410
A.3	总结	419
索引		421

jQuery 入門

Starting with jQuery

如果你正在阅读本页，说明已经从其他人员或者网站论坛中听说过jQuery了，非常希望了解jQuery框架的本质。也许你现在正在使用jQuery，但是想进一步提升技术水平，让老板认可，给你升职加薪。也可能你从没听说过jQuery，只是被封面插图吸引。不论什么原因让你打开本书，以下章节会介绍你全部想知道的知识。

在第一部分仅有的一章里，你会学习到更多关于jQuery框架的知识：它要解决什么问题，为什么要在Web项目中使用它。第1章会告诉你如何选择最合适的jQuery版本。如果你参与了Web网站项目开发，并且想成为一名jQuery高级开发人员，那么请尽情享受本书的学习之旅吧。

本章内容

- jQuery是什么以及为什么使用它?
- 低调的JavaScript战略。
- 选择正确的jQuery版本。
- jQuery框架的基本元素和概念。

“只有两种语言：一种是人们抱怨的，一种是没人使用的”，此话出自C++之父Bjarne Stroustrup之口。这句话也符合JavaScript的现状。这句话同样适用于其他语言（比如PHP），如同这些语言一样，JavaScript多年来一直被标上“坏语言”的标签。然后奇迹出现了，由于Ajax技术的流行，比如Prototype、Moo Tools及jQuery框架的发布，还有一些高度交互的Web应用程序（比如单页应用程序（single-page applications））的出现，让人们重新认识JavaScript的潜力。今天，多亏Node.js（基于V8引擎的网站开发框架）和PhoneGap（开发混合移动APP的框架），JavaScript仍然是使用最广泛的语言之一。

jQuery是免费的（MIT许可证）、流行的JavaScript库，2006年由John Resig创建，设计目标是简化客户端的脚本编程。正如jQuery官方网站所述：

3

jQuery是一个快速、小巧、功能丰富的JavaScript库。jQuery API使得跨浏览器的HTML文档遍历与操作、事件处理、动画、Ajax开发更加方便、简单。基于结合的通用性和可扩展性，jQuery改变了数以百万开发者编写JavaScript的方式。

虽然感觉有点王婆卖瓜，但是这确实是事实。jQuery真的改变了数以百万计的开发者 and 设计者编写自己代码的方式。根据BuiltWith最新的统计报告（2015年4月），jQuery被前100万网站中63%的网站使用(<http://trends.builtwith.com/javascript/jquery>)。最大的竞争对手Moo Tools只有3%的份额(<http://trends.builtwith.com/javascript/MooTools>)，Prototype只有2.5%的份额(<http://trends.builtwith.com/javascript/Prototype>)。

jQuery被世界上最重要公司的网站使用，例如Microsoft、Amazon、Dell、Etsy、Netflix、Best Buy、Instagram、Fox News、GoDaddy等。如果你还有所怀疑，那么这些数据应该可以让你相信jQuery可以用到自己的项目中。

本书从基本的概念开始，比如选择器和DOM文档遍历，到高级知识，比如扩展函数（使用插件）、改进代码性能和测试。本书假定你已经了解了JavaScript基础编程知识。如果需要复习一下知识，那么可以查看附录。如果你是“新手”，则会感到读本书有点吃力，建议你先学习基础知识再来阅读本书。等你回来。

好了吗？很高兴重新见到你！让我们从头开始吧，讨论一下jQuery能带给你什么以及如何帮助你完成Web网站开发。

1.1 写得少，做得多

Write less, do more

jQuery的座右铭是“Write less, do more（写得少，做得多）”。如果你曾经做过向网页中增加动态功能的工作，就会发现使用JavaScript完成时需要很多行代码。jQuery创建的宗旨就是简化这种日常琐碎的代码、易于学习。jQuery还解决了很多浏览器的兼容性问题。

例如，如果使用JavaScript处理radio按钮组，就需要查找哪个radio按钮被选中，然后去获取它的value属性值。也就是首先需要定位radio按钮组，然后通过遍历所有的元素来对比每个元素的checked属性，再获取选中元素的value属性值。

为了兼容IE6以上浏览器（如果忽略更老版本的浏览器，方法会简单很多），其实现代码如下：

```
var checkedValue;
var elements = document.getElementsByTagName('input');
for (var i = 0; i < elements.length; i++) {
    if (elements[i].type === 'radio' &&
        elements[i].name === 'some-radio-group' &&
        elements[i].checked) {
        checkedValue = elements[i].value;
        break;
    }
}
```

与之对比的jQuery实现代码如下：

```
var checkedValue =
    jQuery('input:radio[name="some-radio-group"]:checked').val();
```

不要担心，这些代码看起来有些神秘。很快你就会明白它是如何工作的，编写自己的简洁但强大的代码，用jQuery语句开发出Web网页。现在来讲解一下jQuery如何把多行代码变成一行代码。

jQuery代码如此简洁的原因在于其选择器（selector）和区分页面元素的表达式。它可以使我们轻易定位需要的元素。这个例子中，被选中的元素在radio按钮组里。如果还没有下载示例代码。那么现在是最好的时候。你可以从<http://www.manning.com/derosa>地址下载，然后解压代码。第1章的例子代码路径是chapter-1/radio.group.html。效果如图1-1所示，使用jQuery

来确定哪个radio按钮被选中。

这个例子向你展示了jQuery代码的简洁之道。这并不是jQuery唯一的强大之处。jQuery可以支持复杂的选择器来查询元素，而无须考虑跨浏览器的兼容性问题，特别是旧的浏览器。

进行选择时，需要依赖于两个东西：方法和选择器。现在最新的浏览器都支持原生的查找方法`document.querySelector()`和`document.querySelectorAll()`。允许使用更复杂的选择器来替代这种原始的根据ID或者样式（class）查找的方式。

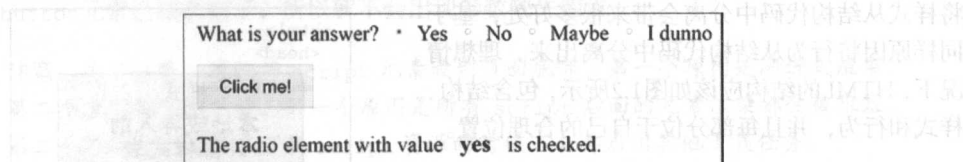


图 1-1 jQuery 用一行代码就可以方便地确认哪个 radio 按钮被选中

5

此外，最新的浏览器都支持CSS3选择器。如果只想支持最新的浏览器，并且只使用jQuery的元素选择功能，那么一定要注意页面jQuery的过度引用。事实上，很多人还在使用旧的浏览器，我们必须考虑这种情况，提供兼容性支持。自己调试兼容性会非常痛苦，这也是为什么使用jQuery的原因。它允许直接使用选择器而不需要考虑浏览器是否支持的问题。

注意：什么是现代浏览器？IE10 以上，最新的 Chrome、Opera、Firefox 和 Safari 都是。

还不相信？这里有一个问题列表，如果不使用jQuery，则需要你自己来处理<http://goo.gl/eULyPT>。此外，正如我们所列举的，jQuery库远比这个强大，你可以在本书的后面看到介绍。

现在来看看如何在网页中使用JavaScript。

1.2 低调的 JavaScript

Unobtrusive JavaScript

没有CSS的时代，需要强制把风格标签和文档结构标签混合在一个HTML页面中。长期开发网页的人绝对不喜欢这种方式。

CSS允许将文档代码和样式定义分离，而无须再定义``这种标签。分离样式不仅让文档易于管理，而且提供了更多的灵活性，可以通过修改不同的CSS引用文件来修改页面的样式。

很少有人愿意使用旧样式和HTML元素混用的风格，但是还是经常可以看到下面的代码：

```
<button onclick="document.getElementById('xyz').style.color='red';">  
Click Me  
</button>
```

可以看到button元素并没有使用这种不推荐的方式定义字体。这取决于页面的CSS规则。虽然这个声明没有混淆style标签和HTML代码，但是它和元素的行为（behavior）混合在一起了。button的onclick事件的处理代码里包括了JavaScript的执行代码来修改DOM元素的样式为红色。让我们看看如何改进这个代码。

1.2.1 分离行为

将样式从结构代码中分离会带来很多好处，基于同样原因将行为从结构代码中分离出来。理想情况下，HTML的结构应该如图1.2所示，包含结构、样式和行为，并且每部分位于自己的合理位置。

这条编程原则称为“低调的JavaScript（unobtrusive JavaScript）（JavaScript位于不显眼的位置，不会引起注意）”，现在被各大主流的JavaScript框架支持，帮助开发人员分离页面代码。jQuery内核已经为这一原则做了优化，以便生成符合规则的代码。“低调的JavaScript”原则认为，任何把JavaScript表达式或者语句放到HTML标签<body>内的做法都是错误的，无论是脚本块放置于页面主体以外的地方还是作为元素的属性（例如onclick）。

你或许会问，“没有onclick属性，怎么给按钮添加事件处理代码？”考虑下面的代码是怎样修改以上例子的：

```
<button id="test-button">Click Me</button>
```

更加简单！但是注意到没有，即使你一刻不停地点一天按钮也不会有结果，因为没有行为处理代码。现在就来完善代码。

1.2.2 分离脚本

我们把JavaScript代码放到单独的script标签中，而不是嵌入页面按钮的代码中。下面就是最佳实践，应该把代码放到页面底部结束标签(</body>)之前：

```
<script>
  document.getElementById('test-button').addEventListener(
    'click',
    function() {
      document.getElementById('xyz').style.color = 'red';
    }
  );
</script>
```

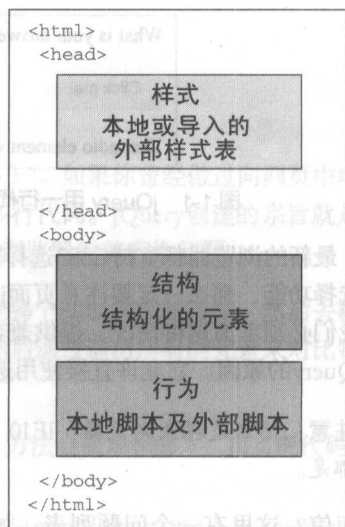


图 1.2 结构、样式和行为分离，可以最大化页面代码的可读性和可维护性

```
    },  
    false  
  );  
</script>
```

因为把代码放到了页面底部，所以不需要使用之前的window对象的onload事件处理方法，或者使用新浏览器支持的DOMContentLoaded事件。DOMContentLoaded事件只在所有页面元素全部加载和解析完成后才会触发，不需要等待样式和图片等资源文件。load事件会等待所有的资源加载完毕才会触发（1.5.3节会详细介绍）。把脚本放到页面底部，当浏览器解析语句时，button元素已经存在了，所以就不会出现参数缺失错误。

7

注意：为了性能，建议将 script 元素放到页面底部。第一个原因是渐进式渲染，第二个原因是并行下载。第一个原因是所有 script 后面的元素渲染都会被阻塞。第二个原因是浏览器在下载 script 资源的时候不会再启用其他下载任务。

上面的例子代码并不能保证百分之百兼容所有的浏览器。因为JavaScript使用了addEventListener()方法，所以IE6至IE8浏览器不支持。本书后面会详细介绍jQuery怎样解决这种问题。

低调的JavaScript，是一种实现分离页面职责的强大技术，但是它并非毫无代价。你或许注意到，为了实现相同的功能，在按钮元素标签里多使用了几行代码。“低调的JavaScript”原则会增加脚本的行数，而且需要良好的客户端脚本编程模式与原则。

但是这些也不是坏事：强迫你编写客户端代码而不是依赖于服务端代码。当然，如果不使用jQuery，就肯定会有额外多余的代码。

jQuery的设计目标就是支持“低调的JavaScript”原则来简化编码工作，无须大量额外的代码工作就能进行开发。你会发现使用jQuery非常高效，编写更少的代码就可以实现更多的功能。这也是jQuery的口号：“写得少，做得多”。事不宜迟，现在就开始看看jQuery如何帮助我们简化开发工作。

1.3 安装 jQuery

Installing jQuery

既然已经知道了jQuery框架的用途，那么现在就可以下载来学习了。可以访问<http://jquery.com/download/>网站进行下载。可能第一次打开页面时会被众多选项吓倒。分支1.x、2.x或3.x？压缩或非压缩版本？下载或者使用内容分发网络（CDN）？选择哪个版本取决于几个因素。为了作出正确选择，下面揭秘这些差别。

8

1.3.1 选择正确版本

2013年4月，为了适应未来Web发展的需求，特别是从浏览器角度考虑设计，jQuery团队正式

引入2.0版本。在此之前，jQuery支持所有最新的Chrome、Firefox、Safari、Opera及IE6以后的版本。2.0版本之后，jQuery团队决定放弃支持IE6、IE7、IE8三个版本，重点关注Web的未来发展需求。

这个决定导致删除了大量代码，这些删除的代码都是为了支持旧版浏览器缺失的特性和兼容性而开发的。这样，框架整体缩小了12%，代码更高效了。虽然1.x和2.x版是两个不同的分支，但是它们之间还是有很强的关联的。jQuery 1.10版和2.0版、1.11版和2.1版等之间具有奇偶校验功能。

2014年10月，jQuery基金会主席Dave Methvin (<https://jquery.org/>) 发表了博客文章(<http://blog.jquery.com/2014/10/29/jquery-3-0-the-next-generations/>)，宣布jQuery将引入新的版本jQuery 3.1.x支持旧的浏览器；2.x版支持新的浏览器；jQuery 3也分为两个版本，jQuery Compat 3是1.x的继任者，jQuery 3是2.x的延伸。他解释的原因如下：

从这个发布开始，将调整关于浏览器的支持策略。jQuery 的主要包仍然紧凑，支持长期存在的浏览器（当前的和之前的版本）。我们或许会根据市场份额来增加对其他浏览器的支持。jQuery Compat 包提供了更多浏览器的支持，但是文件过大，还可能导致性能问题。

对于新版本，jQuery团队放弃了对一些浏览器的支持，修改了很多bug，以及改进了几项功能。

考虑使用哪个版本的首要因素是，项目系统支持什么浏览器。表1.1表示每个jQuery版本支持的不同浏览器。

表 1.1 主要 jQuery 版本支持的浏览器对比

浏览器	jQuery 1	jQuery 2	jQuery Compat 3	jQuery 3
IE	6+	9+	8+	9+
Chrome	当前的和之前的	当前的和之前的	当前的和之前的	当前的和之前的
Firefox	当前的和之前的	当前的和之前的	当前的和之前的	当前的和之前的
Safari	5.1+	5.1+	7.0+	7.0+
Opera	12.1x 当前的和之前的	12.1x 当前的和之前的	当前的和之前的	当前的和之前的
iOS	6.1+	6.1+	7.0+	7.0+
Android	2.3 4.0+	2.3 4.0+	2.3 4.0+	2.3 4.0+

正如大家所看到的，表中浏览器支持的版本有一定程度的重叠。但是，记住“当前的和之前的”的意义不太一样（jQuery新版本发布时，当前的和旧的浏览器版本）。

选择jQuery的另外一个重要考虑因素就是在哪里使用它。下面的一个使用案例可以给大家作参考：

- 不需要支持旧的浏览器，如IE、Opera等时，可以使用3.x版本。这种情况适合控制网站

运行环境的公司局域网。

- 支持尽可能多的用户的公开网站时，使用分支版本1.x。
- 如果网站需要支持广大的用户，但是不需要支持IE6至IE7、Opera和Safari，则应该使用jQuery Compat 3.x版本。
- 如果不需要支持IE8及以下版本，但是需要支持旧的Opera和Safari浏览器，则应该使用jQuery 2.x。
- 使用PhoneGap或者其他类似的框架进行移动APP开发，可以使用3.x版本。
- Firefox OS或者Chrome OS的应用使用jQuery 3.x版本。
- 使用旧插件的网站，看具体的插件代码，可能需要强制使用jQuery 1.x版本。

总之，选择jQuery版本时有两个主要考虑因素：在哪里使用jQuery框架，希望支持什么浏览器。

选择时的另外一个疑虑应该是选择压缩版本（又称minified版本）还是选择非压缩版本。压缩版本主要用在产品阶段，而非压缩版本则用在开发阶段（见图1.3）。压缩后的版本体积小，降低了网络带宽流量。压缩版本主要是删除了无用的空格（缩进符），删除了代码注释，缩短了变量名。压缩后的代码难以读懂，难以调试，这也是为什么要在开发阶段使用非压缩版本的jQuery库的原因，因为压缩版本就是体积小。

◀ 10

非
压
缩
版
本

```
// Handle when the DOM is ready
ready: function( wait ) {

    // Abort if there are pending holds or we're already ready
    if ( wait === true ? --jQuery.readyWait : jQuery.isReady ) {
        return;
    }

    // Make sure body exists, at least, in case IE gets a little overzealous (ticket #5443).
    if ( !document.body ) {
        return setTimeout( jQuery.ready );
    }

    // Remember that the DOM is ready
    jQuery.isReady = true;

    // If a normal DOM Ready event fired, decrement, and wait if need be
    if ( wait !== true && --jQuery.readyWait > 0 ) {
        return;
    }

    // If there are functions bound, to execute
    readyList.resolveWith( document, [ jQuery ] );

    // Trigger any bound ready events
    if ( jQuery.fn.triggerHandler ) {
        jQuery( document ).triggerHandler( "ready" );
        jQuery( document ).off( "ready" );
    }
}
```

压
缩
版
本

```
ready:function(a){if(a===!0?!--m.readyWait:!m.isReady){if(!y.body)return
setTimeout(m.ready);m.isReady=!0,a!==!0&&--m.readyWait>0||(H.resolveWith(y,
[m]),m.fn.triggerHandler&&(m(y).triggerHandler("ready"),m(y).off("ready")));}}
```

图 1.3 jQuery 库里一段非压缩版本的源码和压缩版本的源码

虽然本书以jQuery 1.x为基础来测试支持广泛用户浏览器的代码，但是会引入jQuery 3，并且高亮代码，这样知识体系就可以覆盖最新的知识点。

选择正确版本的jQuery非常重要，而且我们也列举了自托管jQuery和使用CDN的差别。

1.3.2 使用 CDN 改善性能

如今使用CDN服务器来改进网站性能已经是普遍的做法，可用于存储图片或者其他库文件。CDN (content delivery network) 是分布式服务器，可提供高可用性和高性能。你可能知道浏览器需要从网站服务器上同时下载一些固定的内容，通常包含4~6个文件。因为CDN使用了不同的主机Host，所以可以加速整个下载的过程，增加同时下载的文件数量。现在，许多网站都使用了CDN，所以很可能需要的库已经在你的浏览器缓存里了。使用CDN下载jQuery并不能保证所有情况下都能提升高性能，因为有很多因素会影响性能。我们的建议是，测试什么配置适合你的特殊情况。

现在有很多可靠的jQuery的CDN服务器，其中最靠谱的还是jQuery CDN(<http://code.jquery.com>)、Google CDN(<https://developers.google.com/speed/libraries/devguide>)和微软的Microsoft CDN(<http://www.asp.net/ajaxlibrary/cdn.ashx>)。¹

现在假设你使用压缩版的jQuery 1.11.3，并且使用jQuery CDN，则页面的配置代码如下：

```
<script src="//code.jquery.com/jquery-1.11.3.min.js"></script>
```

正如你所看到的，这个代码没有配置下载协议(HTTP或者HTTPS)。反之，也可以指定下载协议。但是请记住，在没有运行在Web Server里的页面中使用这种代码可能会出错。

使用CDN并不能保证万无一失。没有服务器和网络是百分之百在线的，CDN也不例外。当依赖CDN加载jQuery文件时，如果用户浏览器没有缓存文件，或者CDN服务器不可用，那么网站代码可能会停止工作。有些网站可能会带来现实问题。为了避免这个问题，有一种简单的方案可被许多开发者采用，那就是再次引入压缩版的jQuery 1.11.3，那么现在你也会使用这种智能的解决方案了。

```
<script src="//code.jquery.com/jquery-1.11.3.min.js"></script>
<script>window.jQuery || document.write('<script src="javascript/jquery-
1.11.3.min.js"></script>');</script>
```

以上代码背后的玄机就是浏览器从CDN服务器请求jQuery库的拷贝文件，先检查是否已经下载，再测试window对象的jQuery属性是否定义。如果失败，就会下载本地主机的拷贝文件。在这个例子中，jQuery文件存在本地网站的javascript文件夹里。如果jQuery属性存在，就可以安全地使用jQuery方法，而无需下载本地网站的拷贝文件。

¹ 由于某些原因，国外的服务器无法保证可用性。国内用户建议使用国内免费的jQuery CDN服务，如百度或者新浪微博的CDN服务，但是请注意版本。——译者注

可以测试jQuery属性,因为一旦jQuery库加载完毕,就会自动为window对象添加jQuery属性。你可以找到所有库的方法和属性。在开发过程中,建议使用本地的拷贝,以避免任何网络连接问题。

除jQuery属性外,还会发现\$美元符,本书中将大量出现这个符号。尽管看起来有点奇怪,但在JavaScript中它是一个变量或者属性。\$实际上就是jQuery的简称,为替代符号。下面这条语句是源码中的:

```
window.jQuery = window.$ = jQuery;
```

到目前为止,已经学习了如何在网页中包含jQuery库,但是还不知道它的结构。将在第1.4节介绍这个主题。

12

1.4 jQuery 结构

How jQuery is structured

jQuery代码库托管在GitHub(<https://github.com/jquery/jquery>)上,它是记录过去多年前端开发变化发展最完美的例子。虽然并不是严格与jQuery库使用关联,但是知道开发专家如何组织其工作流和开发工具也十分重要。

如果你是有经验的前端开发者,就可能已经看到了机会,但如果是新手,则要花点时间才能看到机会。现在来看,开发团队采用了最新、最酷的技术来开发jQuery框架。其特性如下:

- 基于Chrome的JavaScript引擎构建的平台,可以让我们在服务器端运行JavaScript语言。
- Node.js官方的包管理器用来安装像Grunt这样的包。
- 任务执行器可以自动执行常见的任务,比如构建、测试和压缩项目文件。
- 免费的、分布式的版本控制系统用于维护对于代码文件的跟踪,易于在开发者之间互相协作开发。

换句话说, jQuery代码遵守了异步模块定义(AMD)格式。AMD格式建议定义模块,它的依赖项采用异步加载模式。实际开发中,这意味着,即使把jQuery作为一个独特的、单一的块使用,其源代码也会分裂成几个文件(模块),如图1.4所示。它的依赖项文件由管理器管理——这里使用的是RequireJS。

13

为了让你了解模块内部的构造,这里有一些例子:

- ajax: 包含ajax()、get()和post()。
- deprecated: 包含当前的不推荐使用的方法,这些方法还没有删除。模块内部的内容依赖于jQuery的版本。
- effects: 包含支持动画效果的代码,如animate()和slideUp()。

- event: 包含事件处理器方法代码, 如on()和off()。

源代码基于模块组织形式带来的另外一个好处就是: 可以构建只有自己需要的模块的jQuery版本。

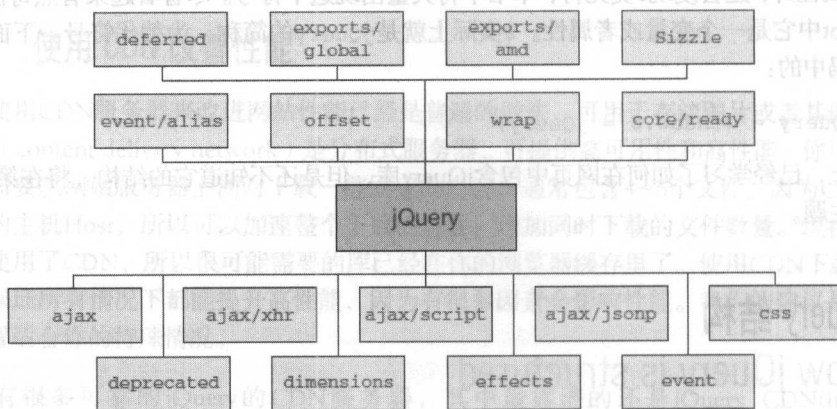


图 1.4 jQuery 模块的架构图: ajax、ajax/xhr、ajax/script、ajax/jsonp、css、deprecated、dimensions、effects、event、event/alias、offset、wrap、core/ready、deferred、exports/global、exports/amd 及 Sizzle

节约空间创建自定义构建库

jQuery 提供了构建自定义只包含自己需要 jQuery 框架功能的可能性。这样可以缩减代码库的大小, 可以改进性能, 因为用户只需要下载很小的文件。

删除多余代码模块非常重要。虽然你可能自认为后来需要 jQuery 全部的功能, 但是在一个网站里使用所有的功能是值得怀疑的。为什么不删除这些不需要的代码来改善性能呢?

你可以使用 Grunt 来创建自定义版本。想象一下你只需要压缩版的 jQuery 1.11.3, 包含所有的功能(不包括过去的方法和属性)和特效。为了完成这个任务, 需要安装 Node.js、Git 和 Grunt。安装完毕以后, 还要复制一份本地的 jQuery 代码库, 使用下面的命令行接口(command-line interface, CLI):

```
git clone git://github.com/jquery/jquery.git
```

一旦复制过程完成, 就输入下面两行命令:

```
npm install
grunt custom:-deprecated,-effects
```

做完了! dist 文件夹下就是自定义构建的 jQuery 库, 包含压缩版和非压缩版。

这个方法并非没有问题。第一个问题出现在新版本 jQuery 发布时。第二个问题是网站需要的

功能之前的模块不存在。这种情况下，需要重新执行之前的步骤（通常是命令）来创建自定义版本，包含需要的新方法、修改bug，或者缺失的模块。

现在知道如何下载jQuery代码库，以及如何创建自定义版本了，是时候来深入介绍jQuery的本质了。

14

1.5 jQuery 本质

jQuery fundamentals

jQuery内核专注于HTML页面元素的查找和执行操作。如果熟悉CSS，就应该知道选择器的强大威力，它通过类型、属性、文档位置等来区分元素组。使用jQuery，可以使用这些知识最大限度地简化JavaScript编程工作。

jQuery把浏览器的兼容性放到高优先级的位置，很多头疼的问题已经在背后悄悄解决了。你会发现jQuery容易扩展，只需要通过插件就可以扩展其他的功能，这个高级主题将在第12章详细讨论。

现在开始介绍jQuery对象，以及如何使用CSS知识来生成强大且简洁的代码。

1.5.1 属性、工具和方法

正如之前说的，jQuery库通过jQuery属性暴露，并且简写符号为\$。可以使用它们来访问属性、方法和jQuery提供的函数。

jQuery暴露的属性之一就是fx.off。它允许使用jQuery的方法来启用或者禁用特效。我们会在第9章详细讨论这个属性。

更激动人心的是utilities工具库，也称为utilities函数。可以把它们当成常用的函数库，也可以把jQuery当成它们的命名空间。

为了了解它们的基本概念，先看一个例子。工具库的函数之一就是字符串剪切函数。可以从头到尾删除字符串的空格。可以看到如下代码：

```
var trimmed = $.trim(someString);
```

如果someString的值是“hello”，使用\$.trim()函数处理以后，结果就是“hello”。正如你所看到的，这个例子中使用了jQuery的简写符号(\$)。记住，这只是一个标识符，和其他的JavaScript标识符作用一样。可以使用jQuery编写同样作用的代码，参考如下：

```
var trimmed = jQuery.trim(someString);
```

另一个工具函数的例子是\$.isArray()，你可能已了解，它用于判断输入参数是不是数组的。

除了属性和函数，jQuery类库也提供了通过调用jQuery()函数才可以访问的方法。下面将深入学习。

1.5.2 jQuery 对象

学习jQuery时第一个要学习的函数就是jQuery()，它接受两个参数，依赖参数个数和类型来执行不同的任务。

与jQuery库里的其他方法一样，它允许链式(chaining)调用。链式调用是一种编程技巧，允许在单行代码中调用多个方法。代码如下：

```
var obj = new Obj();
obj.method();
obj.anotherMethod();
obj.yetAnotherMethod();
```

也可以编写以下代码：

```
var obj = new Obj();
obj.method().anotherMethod().yetAnotherMethod();
```

jQuery()最常见的用法就是选择DOM元素，然后修改它们，它接受两个参数：选择器和上下文(可选参数)。这个函数返回匹配条件的DOM对象集合。什么是选择器？

当引入CSS技术来分离设计与内容时，需要一种方法，以一组页面元素的集合来引用外部的样式定义。这种方法就是使用选择器，代表元素的类型、属性或者在HTML文档中的位置。

XML人员应该熟悉XPath(<http://www.w3.org/TR/xpath20/>)，用来选择XML文档中的元素。

CSS选择器也是等价的概念，只是用于HTML页面中，且更简洁，更易于理解。

jQuery使用了与CSS一样的选择器。它不仅支持CSS2.1，还支持CSS3中的定义。

这一点很重要，因为并不是所有的浏览器都支持所有的实现，或者根本就没有支持(例如旧版本的IE)。如果这个选择器无法满足，jQuery还有自己的选择器，而且允许用户自己定义选择器。

本书会先介绍现有的CSS知识，然后介绍学习jQuery支持的更高级的选择器知识。

如果不熟悉，也没有关系，我们会在第2章详细介绍jQuery选择器。你也可以在jQuery官方网站中看到完整的列表(<http://api.jquery.com/category/selectors/>)。

假想使用jQuery()选择页面中的所有<p>元素标签，则语法如下：

```
var paragraphs = jQuery('p');
```

jQuery库会从文档根部来查询所有匹配的DOM元素，因此，如果元素很多，那么查询过程可能很慢。

绝大部分情况可以使用context参数来加速查找，用来限制查找的子树或者依赖的选择器。为了便于大家理解，可修改上面的代码。

加入你想选择<div>中的所有<p>元素。包含（contained）并不意味着<div>是<p>的直接父节点，也可能嵌套了几级节点。可以使用下面的代码来查找：

```
var paragraphsInDiv = jQuery('p', 'div');
```

使用jQuery简化符号，语句如下：

```
var paragraphsInDiv = $('p', 'div');
```

当设置了第二个参数时，jQuery的第一个集合元素就会基于context来查找，然后查找匹配第一个参数的子元素selector。我们将在第2章详细讨论这个问题。

正如我们所说的，jQuery()（简称\$()）函数返回一个JavaScript对象，这个对象集合包含了匹配选择器的DOM元素。这个对象包含许多预定义方法可以处理的集合元素。我们会使用jQuery集合、jQuery对象或jQuery集（或者其他相似的词语）来代表返回的可供jQuery操作的JavaScript对象。根据定义，前一个变量paragraphsInDiv就是包含所有div元素子元素集合的jQuery对象。要执行操作，可以直接调用jQuery对象方法，比如在页面元素上使用动画效果或设置样式。

正如之前提到的，大量的jQuery方法中的一个重要特性就是允许链式调用。在方法结束工作之后，它会返回自己操作的对象元素集合，准备接受下一个操作。一旦需求变得复杂，使用jQuery链式调用就可以大大简化代码的数量。

前一节内容中强调了把JavaScript代码放到页面底部的好处。多年来，开发者已经习惯了把脚本放到<head>中的scripts元素里，并且调用jQuery的ready()方法。现在虽然不提倡这种做法了，但是许多开发者还在使用。下一节会深入学习现在提倡的方式是什么。

1.5.3 文档准备处理器

拥抱低调的JavaScript，行为就与结构分离。遵守这条原则，就可以在文档页面元素的外部来执行操作。为了执行这些操作，就要等到DOM元素全部加载完毕。

在按钮组的示例中，整个页面必须在行为执行前加载完毕。传统上，使用window对象的onload处理器在页面加载完毕后执行代码。语法如下：

```
window.onload = function() {  
    //这里编写功能代码  
};
```

这些代码会在文档全部加载完毕后执行。不幸的是，浏览器不仅会推迟执行onload的代码直到DOM树创建完毕，而且会等到页面所有的外部资源全部加载并显示到浏览器窗口才执行。这里的资源可能包括图片及Flash动画等。因此，用户可能在从看到页面到等待脚本执行之间等待很长时间。

更糟糕的是，如果图片或者其他资源花费了大量的时间，用户就必须等待图片加载完成才能支持其他的操作行为。这可能会让低调的JavaScript成为噩梦。

较好的办法就是等待文档结构解析完毕并且浏览器已经把HTML转换为DOM树，才执行富行为脚本操作。跨旧浏览器支持实现这个功能确实比较困难，但是jQuery已经提供了简单的封装接口，可以直接在DOM树加载完毕后来执行脚本（无须等待外部资源）。

标准语法如下：

```
jQuery(document).ready(function() {  
    //你的代码...  
});
```

首先使用jQuery()包装document对象，然后调用ready()方法，然后传递一个要执行的函数，在准备好后执行。这意味着在ready()方法内部的函数可以安全地访问页面的所有元素。图1.5展示了这种机制。

笔者称它为正式语法是有原因的，更简单的写法如下：

```
jQuery(function() {  
    //你的代码...  
});
```

通过传递函数给jQuery()（或简称\$()），就可以通知浏览器在DOM加载完毕后就执行操作。更好的是，你可以在相同的HTML文档里多次使用它，浏览器加工后执行所有声明的函数。

相反，onload只能执行单个函数。这个局限也导致很多第三方插件如果使用这种做法，那么bug很难调试。

使用document-ready处理器是拥抱低调的JavaScript原则的好方法，但是并不是强制的，也可以避免使用。

因为ready()需要在DOM加载完毕后执行，所以开发者经常把script放到页面<head>标签里。正如第1.2.2节讨论的分离脚本，可以把脚本放到页面关闭body标签(</body>)之前。这时DOM中的所有元素都应该准备完毕了。

因此，可以安全地访问这些元素了。如果想知道如何避免使用\$(document).ready()，则可

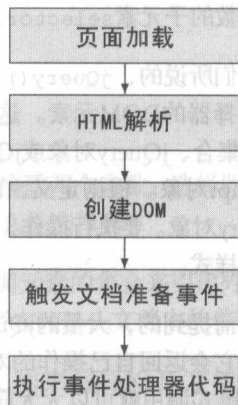


图 1.5 浏览器在文档准备完成后执行的步骤

以看源代码下面的例子chapter-1/radio.group.html。

本书的其余部分会坚持当前的最佳实践，所以无须使用ready()方法。

1.6 总结

Summary

本章介绍了jQuery编程的很多内容。作为总结，jQuery非常有用，适合任何网页进行烦琐的JavaScript编程的情况，而且支持低调的JavaScript原则。使用这个方法可以分离结构代码和行为，正如分离CSS代码和结构一样，使页面组织更加合理，增加了页面代码的重用性。

尽管jQuery在JavaScript命名空间里只引入了两个名字——jQuery函数和它的简称\$，但是，jQuery库提供了大量函数，根据传入参数的不同而执行不同的功能、操作。

本章提到了jQuery代码库的存储和组织结构，也介绍了一些可用的jQuery框架的版本及其差别。性能是最重要的考虑因素之一，因此，也介绍了如何根据页面需要来删减jQuery库。使用CDN和自定义模块方法都是最好的优化页面的方式。

本章还介绍了jQuery提供给Web开发者的功能。第2章会介绍如何使用jQuery选择器来快捷地选择需要执行操作的元素。

19

本书的其余部分会坚持当前的最佳实践，所以无须使用ready()方法。

- jQuery UI：创建界面用交互、特效、控件、小部件。
- jQuery Mobile：面向手机和设备平台开发的基础，跨平台的移动、可移植的网页应用。
- jQuery MVC：进行MVC测试的框架。
- Plugins：jQuery发布在<http://jquery.com/plugins>网站上，由社区开发者和公司的解决方案创建的或者改进功能、插件、控件开发。

阅读第2章，就会知道了解jQuery库，并学习使用最强大的客户端开发工具开发网页，所以通过这一章，就为本书的后续学习jQuery库最有用的开发、测试应用打下了坚实的基础。

20

第二部分

jQuery 核心 Core jQuery

本章内容

- jQuery 的 CSS 选择器选择。
- jQuery 拥有的过滤器。
- jQuery 的过滤器。
- jQuery 的 jQuery() 函数的使用。

从 John Resig 创立 jQuery 框架到现在已经很多年了。但是创立之初的许多年，jQuery 也只是能操作 DOM 的简单框架。之后的许多年，jQuery 已经建立了自己完整的生态系统，包含许多优秀的库和其他项目，名单如下：

- jQuery UI：创建界面 UI 交互、特效、微件、主题库。
- jQuery 移动：所有主流移动设备平台开发的基于 HTML5 的 UI 框架，可帮助我们设计完美的移动 Web 应用。
- Qunit：jQuery 进行单元测试的框架。
- Plugins：Plugins 发布在 npm(<https://www.npmjs.com/>) 网站上，由全球开发者创建的解决不同问题的或者改进功能的库已经传播开来。

阅读第 2 章，就会彻底了解 jQuery 库，并且可以使用最强大的客户端开发工具来开发项目。所以翻过这一页，就挖掘和准备学习如何用简单且有趣的方式为 Web 应用注入鲜活的生命力！

◀ 21

本章内容

- 使用jQuery和CSS选择器选择元素。
- 探索jQuery独有的过滤器。
- 开发自定义过滤器。
- 深入学习jQuery()函数的context函数。

本章会详细讲解如何通过jQuery最强大、最常用的函数jQuery()找到要操作的DOM元素：通过选择器选择DOM元素。通过本章的学习，你会熟悉很多选择器的使用方法。jQuery不仅支持CSS选择器，还引入了其他一些选择器。本章也会介绍过滤器（filters），且很多是jQuery特有的用来进一步过滤匹配结果集合的。如果还不能满足需要，那么可以自定义过滤器（称为自定义选择器或者自定义伪选择器）。本章还会介绍上下文（context）、jQuery()函数的第二个参数，并且讲述如何使用它们。

23

Web应用交互需要的许多功能都是通过操作页面的DOM元素来实现的。但是在操作这些元素之前，需要先区分并选中它们。本章和第3章会讲述选择元素的概念。本书的上一版中，只有一章，以为它们的内容是紧密相关的，但是现在决定分成两章，以便给大家介绍海量的知识点。注意，尽管分为两章，本章依然很长，可能需要多读几遍才能掌握所有的概念。记住最后一个提示，让我们开始学习jQuery是如何选择要操作的目标元素的。

2.1 选择操作元素

Selecting elements for manipulation

使用任何jQuery方法要做的第一件事就是选择要执行操作的元素。正如第1章中介绍的，使用jQuery选择页面元素，需要传递参数给jQuery()函数（或者简称\$()）。jQuery()函数会返回包含匹配过滤条件的DOM元素的集合，并且结果集支持许多jQuery方法和属性。

有时候要选择的元素很好描述，比如“所有的页面元素”，有时候就比较复杂，比如“所有使用了样式list-element、包含link链接元素且是列表中的第一个元素”。幸运的是，jQuery提供了强大的选择器语法，可以让我们优雅、简洁地来区分特定的元素集合。你可能已经熟悉了

很多语法。jQuery使用了我们熟知的CSS语法，并且为一些公共和复杂的操作进行了扩展。



为了帮助用户学习元素选择，我们把很多例子放到了jQuery选择器动手实验室页面(文件位置：chapter-2/lab.selectors.html)。这个页面允许大家输入jQuery选择器字符串，实时观察哪个DOM元素被选中，如图2.1所示。

jQuery Selectors Lab Page

Selector Panel

Type a selector into the text field below and click the Apply button.


Selector:

jQuery statement:

0 matching element(s):

Dom Sample

Some images:



This is a <div> with an id of someDiv

Hello, I'm a <h2> element

I'm a paragraph, nice to meet you.

- jQuery website
 - CSS1
 - CSS2
 - CSS3
 - Basic XPath
- jQuery also supports
 - Custom selectors
 - Form selectors

Language	Type	Invented
Java	Static	1995
Ruby	Dynamic	1993
Smalltalk	Dynamic	1972
C++	Static	1983

Text:

Radio group: ☐ A ☒ B ☐ C

Checkboxes: ☐ 1 ☐ 2 ☒ 3 ☐ 4

Dom Sample code

```
<span>Some images:</span>
<div>
  
  
  
  
  
  
</div>

<div id="someDiv">This is a <div> with an id of <code>someDiv</code></div>
```

图 2.1 jQuery 选择器试验页面允许你实时观察输入的任何选择器

提示：如果还没有下载例子代码，现在就应该下载了——如果跟着做练习，就更容易吸收本章的知识。请访问 <http://www.manning.com/derosa> 页面或者 <https://github.com/AurelioDeRosa/jquery-in-action> 页面下载代码。

左上角的选择器面板 (Selector Panel) 包含一个文本框和按钮。要进行试验操作，就在输入框

输入选择器，然后单击按钮。现在输入一个字符串li，然后单击“Apply”按钮。

输入的选择器（假如是li）被应用到了HTML元素中，加载到右上角的“Dom Sample”框里。这个试验会在你输入class名字found-element并单击“Apply”按钮后执行匹配操作。为本网页定义的CSS声明会把所有的匹配元素高亮显示在黑边和灰色背景框里。单击“Apply”按钮后，应该可以看到图2.2所示的效果，所有的li元素都被高亮显示。此外，执行jQuery语句，选中的元素名称也会限制在输入框下面，标记了“DOM Sample Code”。这样可以帮你编写选择器选中特定的目标元素。

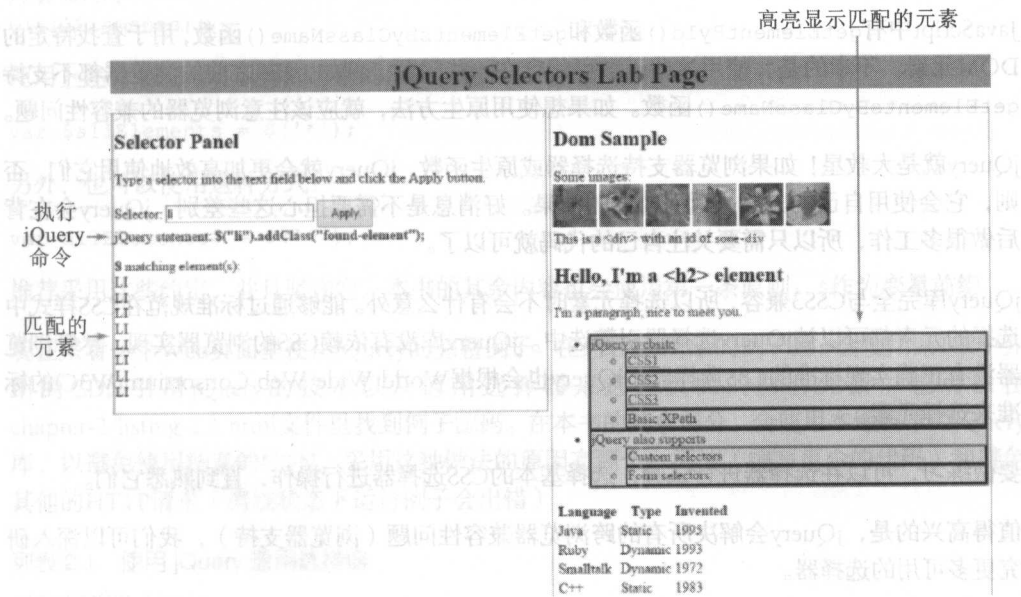


图 2.2 选择器值 li 匹配的所有 li 元素

随着本章内容的深入，会介绍更多关于如何使用试验页面。现在看看jQuery如何处理基本的CSS选择器。

2.2 基本选择器

Basic selectors

为了给页面元素使用样式，Web开发者需要熟悉一些跨浏览器使用的简单的选择表达式语法。这些表达式通过元素ID、CSS样式名字或者tag名字来选择元素。通过tag名字来选择元素的例子就是通用选择器，它允许选择DOM中的所有页面元素。这个选择表达式允许在DOM中执行简单的查询操作，并且会在下面详细讲解。组合使用可以进行复杂的查询。表2.1列举了几个例子及其组合。

表 2.1 简单的 CSS 选择器例子

例 子	描 述	在 CSS 中
*	匹配页面所有元素	✓
#special-id	匹配 ID 为 special-id 的元素	✓
.special-class	匹配样式为 special-class 的元素	✓
a	匹配 a 标签	✓
a.special-class	匹配所有样式为 special-class 的 a 标签	✓
.class.special-class	匹配使用了样式 class 和 special-class 的元素	✓

JavaScript中有`getElementById()`函数和`getElementsByName()`函数,用于查找特定的DOM元素。不幸的是,使用这些函数可能存在一些问题。例如,IE9之前的浏览器都不支持`getElementsByName()`函数。如果想使用原生方法,就应该注意浏览器的兼容性问题。

jQuery就是大救星!如果浏览器支持选择器或原生函数,jQuery就会更加高效地使用它们,否则,它会使用自己的方法来返回需要的结果。好消息是不需要担心这些差别。jQuery会在背后做很多工作,所以只需要关注自己的代码就可以了。

jQuery库完全与CSS3兼容,所以选择元素时不会有什么意外。能够通过标准规范在CSS样式中选择的元素都可以被jQuery选择器引擎选中。jQuery库没有依赖CSS的浏览器实现。尽管浏览器没有正确实现标准的CSS选择器,jQuery也会根据World Wide Web Consortium(W3C)的标准来选择元素。



要做练习,可以在选择器进行试验,选择基本的CSS选择器进行操作,直到熟悉它们。

值得高兴的是,jQuery会解决所有的跨浏览器兼容性问题(浏览器支持),我们可以深入研究更多可用的选择器。

2.2.1 通用选择器

第一个可用的选择器就是通用选择器,它可以选择所有的元素,代表符号是星号(*),允许选择页面的所有元素,甚至包括head元素及其子元素。为了强化这个概念,让我们看看下面的HTML代码:

```

<!DOCTYPE html>
<html>
  <head>
    <title>jQuery in Action, 3rd edition</title>
  </head>
  <body>
    <p>I'm a paragraph</p>
  </body>
</html>

```

选择页面的所有元素时,需要使用通用选择器传递给`jQuery()`函数(或者简称`$()`),代码

如下：

```
var allElements = $('*');
```

继续介绍之前，还要提一下命名约定。当调用jQuery保存结果集到变量中时，广泛采用的约定是前缀或者后缀的一个美元符。这个并没有特殊含义，只是提示变量存储的数据。另外，确保不会再次在DOM元素上调用`$()`函数，因为已经调用一次了。例如，可以像下面这样编写代码：

```
var allElements = $('*');  
//其他代码  
$(allElements);
```

使用上述命名规则可以拆卸之前的代码，使用前置美元符`$`，代码如下：

```
var $allElements = $('*');
```

另外，也可以使用这种方式：

```
var allElements$ = $('*');
```

推荐采用这些约定，并且坚守它。本书的其余内容也会使用第一条原则：`$`作为变量前缀。

现在来看一个Web页面里使用jQuery的完整例子。在列表2.1所示的例子中，会使用第1章中介绍的CDN引用jQuery的技术以及通用选择器来选择页面的所有元素。也可以在chapter-2/listing-2.1.html文件里找到例子源码。在本书的其余部分，会使用本地版本的jQuery库，以避免使用任意的CDN。采用这种做法的原因有两个：简洁（编写更少的代码）和避免其他的HTTP请求（离线状态下运行例子会出错）。

列表 2.1 使用 jQuery 通用选择器

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>jQuery in Action, 3rd edition</title>  
  </head>  
  <body>  
    <p>I'm a paragraph</p>  
    <script src="//code.jquery.com/jquery-1.11.3.min.js"></script>  
    <script>  
      window.jQuery || document.write('<script src="../../js/jquery-  
1.11.3.min.js"></script>');  
      var $allElements = $('*');  
    </script>  
  </body>  
</html>
```

从 jQuery CDN 请求 jQuery

在页面中选择所有元素

如果 CDN 无法访问，就使用本地文件

28

我们介绍过上面的列表会宣传所有的页面元素，但是这些元素包含什么？如果使用调试工具或者控制台（浏览器可用）来监控变量，就会看到html、head、title、body、p、script（页面第一个）和script（页面第二个）。

警告：要指出的是，旧的浏览器(IE6~IE7)并不支持 `console.log()` 方法。本书的例子中会忽略这个问题，会经常使用这个方法避免使用 `window.alert()` 方法。

但是，应该记住某些浏览器可能不支持所需要的功能。

记住，查找和存储的元素的顺序与页面出现的顺序一样。

开发工具

无调试工具开发DOM脚本应用程序就好像带着婚纱手套弹钢琴。为什么要这么做呢？因为大家使用的浏览器有许多不同的方式查看代码。所有新的浏览器都内置了这种工具，虽然名字不同，但都可以正常使用。例如，在Chrome浏览器里，工具叫做Developer Tools (<https://developers.google.com/chrome-developer-tools/>)，而在IE里叫做Developer Tools (<https://developers.google.com/chrome-developer-tools/>)。Firefox也有自己内置的调试工具，但是开发者通常把这个插件叫做Firebug (<http://getfirebug.com>)。这个工具不仅允许我们检查JavaScript控制台，也允许我们实时监测DOM、CSS、脚本以及开发过程中页面的其他状态数据。

正如你所看到的，使用通用选择器会强迫jQuery遍历所有的DOM元素节点。如果DOM中的元素很多，那么处理过程就会很慢。因此，这种用法不提倡。此外，要查询所有元素的可能性也不高，大部分情况都是查询某个元素的子元素。这一点后面会看到。

如果已经使用过JavaScript和浏览器编程，就会知道大部分选择都是通过ID来查找元素的。下面来深入这个主题。

2.2.2 ID 选择器

ID选择器是最常用的选择器，不仅在jQuery中最常用的，而且在原始的JavaScript编程也是最常用的。在JavaScript中，`document.getElementById()` 函数使用ID选择元素。如果了解CSS，就会想起ID选择器在元素ID之前使用了一个#符号（在有些国家，这个符号的意义不同，有的是数字符号，有的是英镑符号）。如果把这段代码放到页面里，

```
<p id="description">jQuery in Action is a book about jQuery</p>
```

就可以通过下面的代码使用ID来查找元素：

```
$('#description');
```

当使用ID选择器时，jQuery会返回零个或者一个DOM元素。如果页面中有多个元素使用了相同的ID，那么jQuery也只会返回第一个元素。尽管可以设置多个元素使用同样的ID，但是这样是无效的，尽量不要这样做。

注意：W3C的HTML5规范断言，“ID的值必须是不包含空格的字符串。对于ID，

就没有其他严格的限制了；尤其是，ID 可以只有数字、以数字开始、以下划线开始、只有标点等”。可能会使用圆点这种符号(.)，它们在 CSS 里有特殊的用法，jQuery 会遵守 CSS 的语法。因此，如果要特殊处理 ID 为 .description 的元素，就要使用到圆点(.)，必须在 jQuery 中加入两根前斜线，语法为 \$('#\\.description')。

在本节开始的时候对比 jQuery 使用 ID 和 JavaScript 如何通过 getElementById() 函数来查找元素一点都不意外。无论使用什么浏览器，jQuery 使用 ID 查询是速度最快的。因为 jQuery 库使用的就是 getElementById() 函数，它本身就很快。

使用 ID 选择器可以快速选择 DOM 中的元素。很多时候，也需要根据 class 样式名称来选择元素。那应该怎么做呢？

2.2.3 Class 选择器

Class 选择器使用 CSS 样式 class 名称来查询元素。作为 JavaScript 开发者，应该熟悉原生的 JavaScript 函数 getElementsByName()。jQuery 遵循这个约定，所以可以在 class 名称之前加上一个圆点。例如，如果在页面 <body> 中定义如下 HTML 代码：

```
<div>
  <h1 class="green">A title</h1>
  <p class="description">I'm a paragraph</p>
</div>
<div>
  <h1 class="green">Another title</h1>
  <p class="description blue">I'm yet another paragraph</p>
</div>
```

30

如果想查询使用了样式 description 的元素，就需要传递 .description 参数给 \$() 函数，代码如下：

```
var $descriptions = $('.description');
```

返回的结果是一个对象，通常作为 jQuery 对象或者 jQuery 集合，它包含两段 HTML 代码。jQuery 库也可以选择使用多个样式的元素，像第二个元素。

在 jQuery 中，和 CSS 一样，可以联合使用多个 CSS 名字选择器。如果想选择使用样式 description 和 blue 的元素，则可以连接查询语句，如 \$('.description.blue')。

Class 样式选择器在 JavaScript 和 CSS 中也是最常用的选择器，但还有另外一种基本的选择器，下面进行介绍。

2.2.4 元素选择器

元素选择器允许通过元素的名称来查询它们。jQuery 使用 getElementsByTagName() 函数来查

找元素。绝大部分浏览器（包括IE6）都支持这个函数。为了了解可以使用元素选择器执行何种查询，可假想查找页面中的所有<div>元素。为了实现这个目标，必须编写如下代码：

```
var $divs = $('div');
```

常见的用法是和别的选择器一起使用，因为页面通常都有许多相同类型的元素。比如，必须要在其他选择器前写上元素选择器。又如，如果想查找所有使用了class样式clearfix的元素，那么语句如下：

```
var $clearfixDivs = $('div.clearfix');
```

虽然也可以和ID选择器一起使用，但是我们极其反对这样做，原因有两个：性能和无用。使用了复杂的选择器，jQuery会使用自己的方法来执行查询，通常会避免使用原生函数，否则会导致低性能查询。此外，正如在ID选择器一节中指出的，jQuery会查询到第一个匹配ID的元素。因此，如果只查询一个元素，那么没有必要使用两种类型的选择器。

jQuery也允许在单个选择时使用不同类型的元素标签，以优化性能，因为DOM遍历只有一次。要使用这种语法，必须在选择器之间加个逗号（逗号后面的空格会被忽略）。比如，为了选

31 择页面中的<div>和，则编写如下语句：

```
$('div,span');
```

一旦有多于一个元素匹配了逗号分隔的选择器（其实不可能，因为元素只有一种类型div或span），jQuery库就会只查询一次，删除所有的重复项。

由于学习这些上面介绍的几种选择器，可以在DOM里执行基本的元素查询。但是通常我们需要用更复杂的条件来查找元素。

可能需要根据其他节点来查找DOM节点，例如，查询无序的li元素中的所有连接元素时，要做的就是根据元素的层次来指定查询条件。如何执行这种查询，第2.3节中会详细介绍。

2.3 根据层级查找元素

Retrieving elements by their hierarchy

通过样式名称来查找元素这个功能不错，但是通常不需要查询整个页面。有时可能只需要查找某个元素的子元素。思考下面的选择器试验页面DOM文档例子的HTML代码片段：

```
<ul class="my-list">
  <li>
    <a href="http://jquery.com">jQuery supports</a>
    <ul>
      <li><a href="css1">CSS1</a></li>
      <li><a href="css2">CSS2</a></li>
      <li><a href="css3">CSS3</a></li>
      <li>Basic XPath</li>
    </ul>
  </li>
</ul>
```



```

</li>
<li>jQuery also supports
  <ul>
    <li>Custom selectors</li>
    <li>Form selectors</li>
  </ul>
</li>
</ul>

```

假设想选择指向jQuery官方网站的a标签而不是页面中所有的a元素，就可以使用子选择器（Child selector），父元素和子元素通过左尖括号(>)连接。代码如下：

```
ul.my-list > li > a
```

这个选择器只会选择li元素的直接子元素，它们位于元素内并且使用了样式my-list。包含子列表的ul被执行查询，因为它们的父节点没有使用样式my-list。在试验页面运行这个选择器，结果如图2.3所示。

32

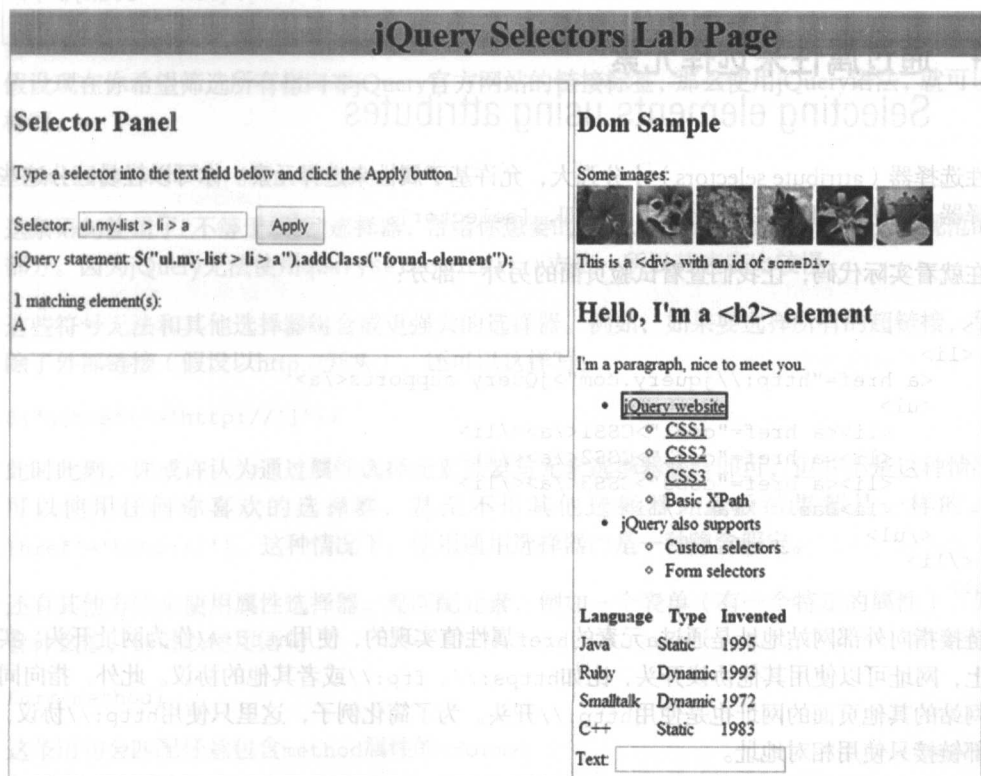


图 2.3 使用 ul.my-list>li>a 选择器，只返回匹配父节点的直接子元素

子元素选择器并不是只能表示基于DOM树层次的两两或者多者之间的关系。表2.2提供了这种选择器的概要说明。

表 2.2 jQuery 支持的 CSS 层次选择器

选择器	描述	在 CSS 中
E F	匹配标签名为 F，且属于 E 的子元素的所有元素	✓
E>F	匹配标签名为 F，且属于 E 的直接子元素的所有元素	✓
E+F	匹配标签名为 F，且在紧邻同级元素 E 之前	✓
E~F	匹配标签名为 F，且在任意的同级元素 E 之前	✓

表 2.2 中所有的选择器，只有第一个是 CSS2.1 规范的一部分，因此它们都不被 IE6 支持。但是，33 可以在 jQuery 中安全地使用它们，因为 jQuery 库已经在底层处理了这些问题。

这些选择器改善了精确查找目标 DOM 元素节点的功能。随着时间的推移，许多新的 CSS 选择器被创建出来以满足其他的需求。其中一个功能就是根据元素的属性 (attribute) 来查找元素。这些新的选择器在第 2.4 节介绍。

2.4 通过属性来选择元素

Selecting elements using attributes

属性选择器 (attribute selectors) 十分强大，允许基于属性来选择元素。你可以轻易区分这些选择器，因为它们都用方括号包围。例如，[selector]。

现在就看实际代码，让我们查看试验页面的另外一部分：

```
<ul>
  <li>
    <a href="http://jquery.com">jQuery supports</a>
    <ul>
      <li><a href="css1">CSS1</a></li>
      <li><a href="css2">CSS2</a></li>
      <li><a href="css3">CSS3</a></li>
      <li>Basic XPath</li>
    </ul>
  </li>
</ul>
```

超链接指向外部网站地址是通过 a 元素的 href 属性值实现的，使用 http:// 作为网址开头。实际上，网址可以使用其他协议开头，比如 https://、ftp:// 或者其他的协议。此外，指向同一网站的其他页面的网址也是使用 http:// 开头。为了简化例子，这里只使用 http:// 协议，内部链接只使用相对地址。

在 CSS 中，可以选择包含 href 值为 http:// 开头的选择器：

```
a[href^='http://']
```

使用 jQuery，语句如下：

```
var $externalLinks = $("a[href^='http://']");
```

这条语句会匹配所有href以http://开头的超链接元素。脱字符（caret character, ^）指示匹配元素必须出现在字符串开头。因为这是正规表达式处理字符串的标准语法，应该容易记忆。



重新打开试验页面，输入[a\[href^='http://'\]](#)，单击“Apply”按钮。注意，只有jQuery链接是高亮的。

34

单引号与双引号

当使用属性选择器时，一定要注意单引号与双引号。错误使用它们可能导致无效的语句。如果样式代码使用了双引号，而且也希望使用它们来包装相同的属性值，就必须避免使用它们。如果认为没有这些字符就难以阅读，那么可以混用两种引号。使用[a\[href^='http://'\]](#)的几种等价语句如下：

```
$( "a[href^=\"http://\"]" );
$( 'a[href^=\'http://\']' );
$( "a[href^='http://']" );
$( 'a[href^="http://"]' );
```

假设现在你希望筛选所有指向非jQuery官方网站的链接标签，那么使用jQuery语法，就可以这样写：

```
$( "a[href!='http://jquery.com']" )
```

这条语句使用了“不等于属性”选择器，带给你想要的结果，因为这个选择器不是CSS规范的一部分。因为jQuery无法使用querySelectorAll()方法，所以其速度比较慢。

这些符号无法和其他选择器组合成更强大的选择器。例如，如果要选择所有的超链接，那么除了外部链接（假设以http://开头），还可以这样写：

```
$( "a[href!^='http://']" );
```

此时此刻，你或许认为通过属性选择元素只要与元素选择器联合即可，但并不是这种情况。可以使用任何你喜欢的选择器，甚至不用其他选择器，最后结果都是一样的，如[\[href^='http://'\]](#)。这种情况下，使用通用选择器(*)是一种隐含假定。

还有其它方法来使用属性选择器。要匹配元素，例如一个表单（有一个特定的属性），则不管什么值，都可以使用语句

```
form[method]
```

这条语句会匹配任意包含method属性的<form>。

为了匹配特定的值，可以使用下面语句：

```
input[type='text']
```

这个选择器会匹配所有类型为text的input元素。

你已经查看了“在开始匹配属性”的选择器的实际代码。这里是另外一个例子：

```
div[title^='my']
```

35 这个选择器会选择所有的div元素，标题的值以my开头。

那怎么选择一个属性以特性的字符结尾呢？用下面的语句：

```
a[href$='.pdf']
```

这个选择器可以筛选所有指向PDF文件的所有链接。

下面这个选择器叫“属性包含（attribute contains）”，可以定位属性里包含这个值的任意元素。它不限制位置：

```
a[href*='jquery.com']
```

正如所期望的，这个选择器会匹配所有包含jQuery网站的元素。

另外一个选择器是“包含前缀（contain prefix）”，它会选中所有属性以给定值开头的元素。如果代码是：

```
div[class|='main']
```

则这个选择器会找到所有的class="main"的div元素，以及所有的class名以main-开头的div元素，比如class="main-footer"。

最后讨论的选择器和上一个相似，只是它仅用来搜索属性值中的特定单词。假设你在使用HTML5的data-*属性，例如data-technologies来指定中的一个列表值，则可以使用下面的选择器来执行选择：

```
span[data-technologies~='javascript']
```

虽然选择的包含这样的属性data-technologies="javascript"，但是也包含data-technologies="jquery javascript qunit"属性。可以把它们当成与样式选择器等价的选择器，但是只针对通用属性。

假如要选择匹配多个条件的节点，选择器可以链式执行。可以在调用链上增加尽可能多的选择器，数量没有固定的限制。例如，可以编写如下：

```
input[type="text"][required]
```

这个选择器会查找所有必备的<input>元素（required属性在HTML5中引入），属于text类型。

表2.3总结了jQuery中可以用于处理属性的CSS选择器。



掌握这些知识后，可以在jQuery动手试验页面做练习，以熟悉表2.3中不同类型选择器的使用。尝试input的其他类型，比如checkbox，值等于1（提示：需要联合几个选择器才能实现）。

选择器不是只能给`$()`函数使用。本章后面你会发现，它们是jQuery方法最常用的参数之一。例如，一旦做了决定，就可以使用jQuery方法和新的选择器来添加一个新元素到之前的集合或者过滤一些元素。

表 2.3 jQuery 支持的属性选择器

选 择 器	描 述	在 CSS 中
<code>E[A]</code>	匹配所有标签名为 E、具有属性 A 的所有元素	✓
<code>E[A='V']</code>	匹配所有标签名为 E、具有属性 A，且 A 的值为 V 的所有元素	✓
<code>E[A^='V']</code>	匹配所有标签名为 E、具有属性 A，且 A 的值以 V 开头的 所有元素	✓
<code>E[A\$='V']</code>	匹配所有标签名为 E、具有属性 A，且 A 的值以 V 结束的 所有元素	✓
<code>E[A!='V']</code>	匹配所有标签名为 E、具有属性 A，且 A 的值不等于 V 的所有元素，或者完全不具备属性 A	✓
<code>E[A*='V']</code>	匹配所有标签名为 E、具有属性 A，且 A 的值包含 V 的所 有元素	✓
<code>E[A = 'V']</code>	匹配所有标签名为 E、具有属性 A，且 A 的值等于 V 或者 以 V-开头的元素	✓
<code>E[A~='V']</code>	匹配所有标签名为 E、具有属性 A，且 A 的值等于 V 或者 包含 V 的所有元素	✓
<code>E[C1][C2]</code>	匹配所有标签名为 E、具有属性 A，且 A 的值满足条件的 C1 和 C2 的所有元素	✓

另外一种情况是，可以查找之前匹配了选择器的集合的全部子节点。

如果介绍的这些选择器还不够用，那么还有更多其他选项来精确地分隔DOM元素。在第2.5节里将介绍其他类型的选择器，名为“过滤器（filters）”。在CSS规范里，这种类型的选择器称为伪样式类型（pseudo-classes）。

2.5 过滤器介绍

Introducing filters

过滤器是与其他类型的选择器结合后进一步筛选匹配元素结果集的选择器。它们非常容易区分，通常以冒号(:)开头。这些选择器的特点之一是，其中一些选择器接受括号内的传递参数，例如，`p:nth-child(2)`。下面将分别讨论jQuery中所有可用的过滤器。

2.5.1 位置过滤器

有时需要根据页面中的位置来选择元素。有时希望选择第一个或最后一个页面中的或第三个段落中的超链接，jQuery支持所有这种特殊需求的查询机制。

例如，考虑编写以下代码：

```
a:first
```

这种格式的选择器会匹配页面上的第一个元素。现在，让我们假设想要选择页面上的第三个元素。为了达到这个目标，可以这样编写代码，如下：

```
a:gt(1)
```

这个选择器很有意思。下面我们来介绍其有趣之处。首先，这个选择器叫“大于(Greater than, gt)”，因为没有选择器叫大于或者等于。另外，与其他目前知道的选择器不同，它接受的参数（这个例子是1）是开始位置的索引。如果希望从第3个开始，为什么要传递1呢？不应该是2吗？答案可以从编程常识里找到，通常索引都是从0开始，第一个元素是0，第二个是1，以此类推。

这些jQuery专用的选择器针对某些棘手的问题提供了优良的解决方案。表2.4列举了jQuery支持的位置过滤器（收藏在jQuery基本过滤器目录下）。

表 2.4 jQuery 支持的位置过滤器

选择器	描 述	在 CSS 中
:first	选择上下文匹配的第一个元素。li a:first 返回列表元素中的第一个 a 标签	
:last	选择上下文匹配的最后一个元素。li a:last 返回列表元素中的最后一个 a 标签	
:even	选择上下文中的偶数元素，li:even 只返回偶数索引元素	
:odd	选择上下文中的奇数元素，li:odd 只返回奇数索引元素	
:eq(n)	选择第 n 个匹配元素	
:gt(n)	大于，选择第 n 个匹配元素之后的元素（排除第 n 个）	
:lt(n)	小于，选择第 n 个匹配元素之前的元素（排除第 n 个）	

正如我们看到的，集合中的元素第一个索引为0。因此，:even选择器将会返回所有奇数位置的元素，因为索引从0开始。例如，:even会返回第一个、第三个，以此类推的所有元素，编号为(0、2等)。:even和:odd的奇偶都是相对于集合中元素的索引来说的，而不是页面位置。

另外一个要强调的知识点就是，可以给:eq()、:gt()、:lt()传递负数作为索引。查询器会按照从最后往前的顺序来计算。如果写了p:gt(-2)，则只选择页面最后一个段落元素p。

考虑到最后一个索引是-1，倒数第二个索引是-2等，事实上，只是想要倒数第二个元素之后

的段落元素p。

有些情况下，我们不想只选择整个页面的第一个或者最后一个元素，而是想要选择相对某个给定父元素的子元素的第一个或者最后一个元素。下面看看如何实现。

2.5.2 子过滤器

我们说过，jQuery支持CSS选择器和规范。因此，可以使用CSS 3中的子伪样式类(pseudo-class)，它们允许根据父元素内部的位置来选择元素。假设想根据给定元素的内部位置来选择内部元素，使用通用选择器。例如，选择父元素的最后一个子元素。

```
ul li:last-child
```

在这个例子中，匹配所有的最后一个子元素。

或许选择给定元素的第四个子元素，则以下代码

```
div p:nth-child(4)
```

会查找所有<div>内部的第四个<p>元素。

:nth-child() 伪样式类与:eq()不同，经常被混淆。前者会计算所有父元素的内部元素，不论这些元素是什么类型。后者只会计算与选择器类型一样的元素。另外一个重要的区别是，:nth-child()继承自CSS规范，因此它假定索引是从1开始而不是从0开始。

另外一个例子：考虑“选择<div>内部具备样式description的所有第二个元素”。这个需求可以使用下面的代码完成：

```
div.description:nth-of-type(2)
```

正如本节学习的，你应该认识到现在可用的选择器非常多且强大。表2.5列举了到目前为止所有jQuery支持的子过滤器。请注意，当一个选择器支持更多语法，如:nthchild()时，会有一个选项：“是否在CSS中？”表2.5意味着支持所有的语法。

表 2.5 jQuery 支持的子过滤器

选 择 器	描 述	在 CSS 中
:first-child	匹配上下文中第一个子元素	✓
:last-child	匹配上下文中最后一个子元素	✓
:first-of-type	匹配给定类型的第一个子元素	✓
:last-of-type	匹配给定类型的最后一个子元素	✓
:nth-child(n)	匹配第 n 个子元素，不论奇数还是偶数，或	✓
:nth-child(even odd)	者匹配基于给定公式、使用给定参数所计算	
:nth-child(Xn+Y)	的第 n 个子元素	

选择器	描 述	在 CSS 中
:nth-last-child(n)	匹配第 n 个子元素，不论奇数还是偶数，或	✓
:nth-last-child(even odd)	者匹配使用给定的参数、基于给定公式计算的	
:nth-last-child(Xn+Y)	从最后一个元素到第一个元素计数的第 n 个子元素	
:nth-of-type(n)	匹配第 n 个子元素，或者匹配具备同样元素	✓
:nth-of-type(even odd)	名的偶数或奇数子元素或其父的第 n 个子元素	
:nth-of-type(Xn+Y)		
:nth-last-of-type(n)	匹配第 n 个子元素，偶数或奇数的子元素；	✓
:nth-last-of-type(even odd)	或其父的具备同样的元素名、从最后一个元素到第一个元素计数的第 n 个子元素	
:nth-last-of-type(Xn+Y)		
:only-child	只匹配没有兄弟元素的子元素	✓
:only-of-type	只匹配没有相同类型的兄弟元素的子元素	✓

如表2.5所示，:nth-child()、:nth-last-child()、:nth-last-of-type()及:nth-of-type()可以接受不同类型的参数。参数可以是索引编号，也可以是偶数，或奇数或等式。公式后面的参数可以是未知的变量，如n。如果想获取多个位置的元素（比如第3、6、9等位置），则可以使用3n表达式。如果要选择任意3倍数后面一位的元素（比如1、4、7等），则可以使用3n+1作为参数。

因为情况越来越复杂，所以下面举例说明。

思考以下DOM中的表格代码，里面列举了一些编程语言和介绍信息：

```
<table id="languages">
  <thead>
    <tr>
      <th>Language</th>
      <th>Type</th>
      <th>Invented</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Java</td>
      <td>Static</td>
      <td>1995</td>
    </tr>
    <tr>
      <td>Ruby</td>
      <td>Dynamic</td>
      <td>1993</td>
    </tr>
    <tr>
      <td>Smalltalk</td>
      <td>Dynamic</td>
      <td>1972</td>
    </tr>
  </tbody>
</table>
```

```

</tr>
<tr>
  <td>C++</td>
  <td>Static</td>
  <td>1983</td>
</tr>
</tbody>
</table>

```

假设你想选择所有包含语言名称的单元，且是每一行的第一个单元格，则可以使用下面的表达式：

```
#languages td:first-child
```

也可以使用：

```
#languages td:nth-child(1)
```

但是第一条语法更加直观、优雅。

要抓取语言类型单元格，则需要修改选择器：`nth-child(2)`，抓取发布时间使用：`nth-child(3)`或者：`last-child`。如果只想获取最后一个单元格（包含1983），则可以使用：`last`这个第2.5.2节介绍过的伪类型，语法是：`td:last`。

要测试功能，可以假想另外一个场景。加上你想查找语言的名称和时间，可以使用：`nth-child()`。

基本上，在这里要查找每个行（`<tr>`）的第一列和第三列（`<td>`）。其实传递一个奇数参数就可以了，语法如下：

```
#languages td:nth-child(odd)
```

要多找点挑战，就再升级前面查找需求的难度，例如要想给：`nth-child()`过滤器传递一个公式。记住，`nth-child()`从1开始，可以把之前的选择器转换成：

```
#languages td:nth-child(2n+1)
```

41

最后一个例子，应该可以让你明白jQuery过滤器的强大之处。



在继续学习下一节内容前，先回到选择器试验页面，尝试选择两个或者四个元素。然后尝试选择三种不同的方式来选择包含“1972”的单元格。感受：`nth-child()`过滤器和绝对位置过滤器之间的差别。

到目前为止，尽管学习的CSS选择器功能已经非常丰富，但还是要学习更多的jQuery选择器，特别是为表单元素设计的过滤器。

2.5.3 表单过滤器

到目前为止，我们看到的CSS选择器都提供了强大的功能和灵活性来匹配目标DOM元素，但

是还有其他更强大的选择器来过滤结果集。

例如，想匹配所有被勾选的复选框的元素，则可以尝试以下代码：

```
$('input[type="checkbox"][checked]');
```

但是，通过属性进行匹配的方式只能检查控件的初始化状态值、HTML标签中最初的元素状态，而我们希望的是实时检查控件的状态。CSS提供了一个伪样式类: checked，可以匹配被勾选的元素。例如，input[type="checkbox"] 语句会选择所有的复选框，而input[type="checkbox"]:checked只会选择是复选框并且被选中的input元素。但是，如果使用过滤器来重写之前的代码，则可以这样：

```
$('input[type="checkbox"]:checked');
```

jQuery也提供了一些强大的自定义过滤器，不通过CSS，标识元素也很简单。例如，:checkbox可以标识所有的复选框元素。结合使用这些自定义选择器，功能会更加强大，而且简化代码。如果只使用过滤器来优化代码，结果简化如下：

```
$('input:checkbox:checked');
```

正如之前讨论的，jQuery支持CSS过滤器选择器，而且可以定义许多自定义选择器，如表2.6所示。

表 2.6 CSS 和自定义的jQuery 过滤器选择器

选 择 器	描 述	在 CSS 中
:checkbox	只选择复选框元素 (input[type=checkbox])	
:checked	只选择状态处于被选中的元素，复选框、单选框或者下拉框元素	✓
:disabled	只选择禁用状态的元素	✓
:enabled	只选择启用状态的元素	✓
:file	只选择文件 input 元素 (input[type=file])	
:focus	只选择处于聚焦状态的元素	✓
:image	只选择图片 input 元素 (input[type=image])	
:input	只选择表单元素 (input、select、textarea、button)	
:password	只选择密码元素 (input[type=password])	
:radio	只选择单选元素 (input[type=radio])	
:reset	只 选 择 重 置 按 钮 (input[type=reset] 或 button[type=reset])	
:selected	只选择处于选中状态的列表元素	
:submit	只 选 择 提 交 按 钮 (button[type=submit] 或者 input[type=submit])	
:text	只选择文本元素 (input[type=text]) 或者没有指定类型的元素 (因为默认是 type=text)	

CSS和自定义jQuery过滤器选择器也可以结合在一起使用。例如，如果想选择所有被启用且处于被选中状态的复选框，则可以使用下面代码：

```
$('input:checkbox:checked:enabled');
```



你可以在试验页面（Selectors Lab Page）里尝试任意过滤器，直到掌握了这些操作为止。

这些过滤器是对之前过滤器的有效补充。但你是不是认为选择器的内容就此结束了？绝对没有！

2.5.4 内容过滤器

在jQuery官方文档里可以看到另外一种类型的过滤器即内容过滤器（content filters）。正如名字的含义一样，这些过滤器是用来根据内容选择的。例如，可以基于某个词语进行选择，或者选择空内容的元素。注意，根据内容并不是说只根据文本内容，也可能是子元素。

正如之前看到的，CSS定义选择器可以选中特定元素的子元素。例如，选择器

43

```
div span
```

将会选择div所有的子元素span。

但是，如果想反着来呢？如果想选择所有包含span元素的<div>元素？这就是:has()过滤器的工作了。考虑下面的选择器：

```
div:has(span)
```

它会选择相对于span的所有div父元素。

当想选择代表复杂结构的元素时，这些机制相当强大。例如，若想根据src属性来查找表元素中某一行包含的图片元素，则可以使用如下的选择器：

```
$('tr:has(img[src="puppy.png"])');
```

它会返回任意符合此条件的子元素行。

完整的内容过滤器的介绍如表2.7所示。

表 2.7 jQuery 支持的内容过滤器

选择器	描述	在 CSS 中
:contains(text)	选择包含特定文本的元素（子元素和文本都会进行评估）	
:empty	选择没有子元素的元素（包括文本）	✓
:has(selector)	选择包含至少有一个元素匹配特定选择器的元素	
:parent	选择包含至少有一个子节点的元素（子节点是文本或者元素）	

如果对这么多选择器或者过滤器感到有困难，建议休息一会，因为内容还没有结束呢！

2.5.5 其他过滤器

我们已经看了许多不可思议的选择器和过滤器（特殊的选择器），有很多可能我们根本不知道它的存在。但是选择器学习之旅尚未结束，下面再学习剩下的选择器。`:visible`和`:hidden`这一对在jQuery官方文档里属于可见性过滤器（visibility filters）目录，但是为了完整性，决定在这里介绍之。

如果想使用过滤器——要选择所有的非复选框的元素，则可以使用`:not()`过滤器。例如，为了选择非复选框元素，可以使用以下语句：

```
input:not(:checkbox)
```

44 但是要注意，它很容易出错，并返回意外的结果！

假设想选择除了`src`属性包含`dog`文本的所有图片，则可以很快想出使用以下选择器的语句：

```
$(':not(img[src*="dog"])');
```

但是，如果使用这个选择器，则会发现不仅获取了所有`src`属性没有引用`dog`的图片元素，而且获取了所有`src`属性包含`dog`的其他非图片类型DOM元素！

哎呀！记住，当忽略基选择器时，它会默认使用通用选择器。选择器代码的实际准确意义是“获取所有的`src`属性没有包含`dog`的非图片元素”，而你想要的是“获取所有的`src`属性没有引用`dog`的图片元素”。此时，选择器代码如下：

```
$('img:not([src*="dog"])');
```



在试验页面里练习，直到熟悉使用`:not()`过滤器选择位置。

当使用jQuery时，页面中隐藏一个或者多个页面元素的情况非常常见。要选择这些隐藏元素，可以使用`:hidden`过滤器。隐藏元素的定义不仅可以包含

```
display:none;
```

的使用，也可以不占用空间。例如，隐藏元素也可以是宽度或者高度设置为0的元素时，使用下面的选择器：

```
input:hidden
```

可以选择页面所有隐藏的`input`元素。

jQuery 3:功能变化

jQuery 3版本修改了`:visible`过滤器的含义(也包括`:hidden`的)。从jQuery 3版本开始，如果元素包含任何布局箱子，包含0宽度或高度，元素就会被认为是`:visible`。例如，`br`元素和没有任何内容的内联元素都会被`:visible`过滤器选中。

当创建页面时，经常会用到外语词汇（笔者是相对于英语来说的）。如果编写了正确语义的HTML页面，则会使用em元素，并且加上lang属性来指定语义。假如你有一个关于比萨（pizza）的页面，可以用下面的标签代码：

```
<p>The first pizza was called <em lang="it">Margherita</em>, and it was  
  created in <em lang="it">Napoli</em> (Italy).</p>
```

可以使用:lang()过滤器来选择页面上的所有外语单词，代码如下：

```
var $foreignWords = $('em:lang(it)');
```

其余过滤器的完整列表如表2.8所示。

表 2.8 jQuery 支持的其余过滤器

选 择 器	描 述	在 CSS 中
:animated	只选择动画模式的元素	
:header	只选择头元素：<h1>到<h6>	
:hidden	只选择隐藏元素	
:lang(language)	只选择采用特定语言的元素	✓
:not(selector)	排除特定选择器元素	✓
:root	选择文档的根元素	✓
:target	通过文档的 URL 框架标识来选择目标元素	✓
:visible	只选择可见的元素	

尽管jQuery提供了许多强大的选择器，但是也无法包含所有的需求场景，而且开发团队也知道这一点。因此，jQuery允许自定义过滤器。让我们看看如何做。

2.5.6 自定义过滤器

第2.5.5节里已经学习了jQuery支持的所有选择器和过滤器。无论它们数量有多少，总有无法处理的情况。也许会发现，使用循环或者选择构造时经常要在同一个结果集上重复执行相同的选择和过滤。这时，可以创建选择DOM节点的快捷方式，换句话说，可以创建自定义过滤器（custom filter）（也称为自定义选择器（custom selector）或自定义伪选择器（custom pseudo-selector））。

jQuery中有两种方法创建自定义过滤器。第一种比较简单，但是不鼓励使用，从jQuery 1.8开始已经被第二种方法取代了。本书只介绍新的方法，当然你也可以通过JS Bin(<http://jsbin.com/ImIboXAz/edit?html,js,console,output>)学习旧的方法。这个例子可在第2章的例子代码chapter-2/custom.filter.old.html中找到。记住，当使用新方法时，你自定义的过滤器在jQuery 1.8之前的版本里不可用。但是，很多情况下这都不是问题，因为这个版本已被淘汰了。

为了解释如何用新方法创建自定义过滤器，就从下面的例子开始。假设要开发一个编程技术

游戏。该游戏有很多级别，可以区分不同的难度，用户可以获取分数，可以获取不同的编程技能。页面代码如下：

```
<ul class="levels">
  <li data-level="1" data-points="1" data-technologies="javascript node
    grunt">Level 1</li>
  <li data-level="2" data-points="10" data-technologies="php composer">Level
    2</li>
  <li data-level="3" data-points="100" data-technologies="jquery
    requirejs">Level 3</li>
  <li data-level="4" data-points="1000" data-technologies="javascript jquery
    backbone">Level 4</li>
</ul>
```

假如经常要选择数据级别(data-level)大于2、获得的数据分数(data-points)大于100的游戏，且包含jQuery技术的(data-technologies)。基于之前学到的技术，你知道如何使用过滤器来查询属性data-technologies包括jquery词语的li元素(li[data-technologies~="jquery"])。但是，如何使用选择器进行大量的对比呢？事实上，不可以。为了完成这个任务，必须循环初始化集合，然后获取需要的元素，代码如下：

```
var $levels = $('> .levels li[data-technologies~="jquery"]');
var matchedLevels = [];
for(var i = 0; i < $levels.length; i++) {
  if ($levels[i].getAttribute('data-level') > 2 &&
      $levels[i].getAttribute('data-points') > 100) {
    matchedLevels.push($levels[i]);
  }
}
```

使用特性选择器初始化选择

在匹配的元素集合上循环迭代

测试当前元素是否匹配

添加到元素集合的末尾

创建自定义过滤器可以不用每次重复以下这几行代码：

```
$>.expr[':'].requiredLevel = $>.expr.createPseudo(function(filterParam) {
  return function(element, context, isXml) {
    return element.getAttribute('data-level') > 2 &&
      element.getAttribute('data-points') > 100;
  };
});
```

测试当前元素

使用 createPseudo() 函数创建过滤器

返回要执行测试的匿名函数

正如你看到的，过滤器只是一个添加了属性:的函数，它属于jQuery的expr属性。它是一个称为“冒号 (colon)”的属性。属性:是一个包含jQuery原生过滤器的属性，可以使用它来添加自己的代码。

你可以调用自定义过滤器requiredLevel，而不是直接传递这个函数，也可以使用称为createPseudo() 的jQuery工具（它属于底层的Sizzle选择器引擎）。


对于createPseudo() 函数，可以传递匿名方法，声明一个参数叫filterParam。filter Param

表示过滤器参数，这个参数也可以另起名字。它是一个可选参数，与`eq()`和`nth-child()`类似。在匿名函数内部，可以创建另外一个匿名函数，用来执行特定的过滤器工作。在函数内，jQuery传递要处理的元素（`element`参数），从这些DOM Element或DOM Document对象（上下文参数）里进行查询。Boolean类型的参数用于指定是不是处理XML文档（`isXML`参数）❶。在最内层的函数里，可以编写代码来测试元素是否被保留❷。在这个例子中，我们可以测试级别是否大于2，用户获得的分数是否大于100。

前一个例子中引入了参数`filterParam`，可以用来向自定义过滤器传递参数。因为需求固定，所以无须使用它。现在就来看看怎么使用它来帮助我们完成工作。

假设要根据提供的分数来查找特定的级别，比如“选择分数大于X的所有级别”。这是一个要使用参数传递给自定义伪选择器的好机会。基于现在的需求，可以创建新的过滤器。

```
$.expr[':'].pointsHigherThan = $.expr.createPseudo(function(filterParam) {  
    var points = parseInt(filterParam, 10);  
    return function(element, context, isXML) {  
        return element.getAttribute('data-points') > points;  
    };  
});
```



和前面的例子有所不同。这里使用的是`createPseudo()`函数，但调用的是`pointsHigherThan`过滤器。在声明第二个函数前，需要在变量`points`❶中保存参数。所以，它可以在闭包里访问（如果不知道什么是闭包，那么可读附录里的相关章节）。此时，可以通过存储的变量来使用给定的参数❷。

让我们来使用新的过滤器。如果想检索所有允许你获得50分以上的级别，那么可以编写代码

```
var $elements = $('li:pointsHigherThan(50)');
```

来获取最后两个项目。

本节中的两个自定义过滤器都可以在`chapter-2/custom.filter.html`下找到代码，也可以在JS Bin(<http://jsbin.com/mucigo/edit?html,js,console,output>)中查看。

48

到目前为止，已经使用了强大的jQuery选择元素函数的一半功力，因为只使用了两个参数中的一个。现在是时候来搞定全部内容了。

2.6 使用上下文增强性能

Enhancing performances using context

在此之前，一直学习的是给jQuery的`$()`函数传递单个参数，这实际上是为了在入门学习阶段简化学习难度。第1章简要介绍了第二个参数`context`。这个参数根据使用的选择器来限制筛选DOM子元素的范围。这个参数在页面有大量元素的情况下非常有用，因为它可以限定

jQuery查询到子树的范围。

正如后面会看到的许多jQuery方法一样，如果有可选参数、忽略参数时，就会使用默认的值替代。这也适用于context参数。当传递第一个参数选择器时，第二个参数context默认为整个document，会应用到DOM树中选择器的每个元素上。

大部分情况下，这也是我们期望的范围，所以采用默认值还不错。但是也有特别的时候，我们希望限定查找范围，而不是查找整个DOM文档。此时，可以使用DOM的子集来限定选择器查找的范围。

针对这种场景，选择器试验（Selectors Lab）页面提供了一个很好的例子。当在页面文本框中输入自定义选择器时，选择器只是应用到DOM例子区域内的DOM子元素集合上。

可以把DOM元素作为context参数，也可以使用包含字符的jQuery选择器或者jQuery结果集合（这意味着可以传递\$()的结果集作为查询上下文，进行二次筛选——不要惊诧，这应该容易理解）。

当选择器或者jQuery集合作为context参数时，筛选的元素会作为第一个参数选择器的查找范围。对于可能包含多个元素的情况，有一个好办法可以分开提供DOM的子树作为查询的上下文。

就拿试验页面作为例子。假设选择器保存在名为selector的变量中。当想使用选择器时，只想把它应用到例子DOM中，它包含在一个ID为sample-dom的DOM元素div中。

假如要这样编写代码调用jQuery函数，

```
$(selector);
```

那么选择器就会应用到整个DOM树上，包括选择器指定的form表单元素。这并不是我们想要的。我们的目标是只处理DOM根元素中ID为sample-dom的div元素，这样就可以来重构代码如下：

49 `$(selector, '#sample-dom');`

它会限制选择器只在DOM的部分子元素中进行查询。

当使用context参数时，jQuery首先会基于这个对象进行搜索，然后选择匹配第一个参数条件的元素。换句话说，查询的匹配选择器的元素都是context的子元素。因此，后代选择器（descendant selector）可以被context取代。

思考下面的查询：选择<div>内部的段落元素<p>。

```
$('div p');
```

现在语句可以修改为：

```
$('p', 'div');
```

二者的查询结果都是一样的。

本节已经完成了对jQuery选择器的学习。现在知道了掌握所有的选择器有多么艰难！但是不要灰心。花点时间来消化这些概念，感觉理解了以后再继续学习。在介绍第3章的方法之前，先做一些练习来测试一下到目前为止已经学习的知识。

2.7 技能测试

Testing your skills with some exercises

本节会针对本章中讲解的选择器和过滤器来做一些练习。如果想测试你的解决方案，那么可以在jQuery选择器试验页面（jQuery Selectors Lab Page）里运行。此外，会提供答案给你作参考对比。

2.7.1 练习

这是练习题的列表。

1. 选择页面的所有超链接元素。
2. 选择一个具备样式wrapper的所有直接超链接<div>子元素。
3. 选择<div>作为上级元素的所有超链接和段落元素。
4. 选择所有data-level等于hard但是data-completed不等于true的元素。
5. 选择所有样式名为wrapper的元素，但是不要使用样式选择器。
6. 在ID为list列表元素内部选择第三个项目，级别任意。
7. 在ID为list列表元素内部选择第二个之后的所有项目(li)。
8. 在样式为description的父元素中，选择序号为3的倍数加1的段落元素（如1、4、7等）。
9. 选择类型为密码（password）的<input>，且是必须的（required属性属于HTML5），属于<form>的第一个元素。
10. 选择页面所有的<div>元素，它们没有子元素、奇数位置（提示：不用索引！），而且没有样式wrapper。
11. 创建自定义过滤器来选择元素，这些元素只包含数字、字母或者下划线作为文本。

◀ 50

2.7.2 解决方案

下面是以上练习题的参考答案。

```
1. $('a')
2. $('div.wrapper>a')
3. $('div a,div p')或者采用更好的办法：使用参数 context$('a,p','div')
4. $('span[data-level="hard"][data-completed!="true"]')
5. $('.[class~="wrapper"]')
6. $('#list li:eq(2)')甚至更好 $('li:eq(2)', '#list')
7. $('li:gt(1)', '#list')
8. $('p.description:nth-child(3n+1)')
9. $('input[required]:password:first-child', 'form')
10. $('div.empty:even:not(.wrapper)')
11. $.expr[":"].onlyText=$.expr.createPseudo(function(filterParam){
    return function(element,context,isXml){
        return element.innerHTML.match(/^\w+$/);
    }
});
```

你是怎么做的？是不是理解了以上这些解决办法？不错！这一节已经完成对所有可用选择器，以及如何创建自定义选择器的回顾。

2.8 总结

Summary

本章专注于使用jQuery提供的区分HTML页面元素的多种方法来创建和筛选自定义结果集（jQuery集合或者匹配元素的集合）。

jQuery提供了许多功能强大的选择器，以支持模式化的CSS选择器，使用准确但强大的语句来查找页面中的元素。这些CSS 3支持的选择器被大部分新的浏览器支持，jQuery不仅支持所有的CSS选择器，还进行扩展，提供了更多强大的选择器来查询页面元素。如果还不能满足需求，那么由于jQuery框架扩展性强大，也非常灵活，因此还允许创建自定义过滤器。

对本章介绍的所有选择器jQuery都提供支持。第3章里将介绍如何使用`$()`函数来创建新的

51 HTML元素。你会看到一些接受选择器作为参数在匹配的结果集上执行操作的方法。

操作jQuery集合

Operating on a jQuery collection

本章内容

- 在DOM中注入新的HTML元素。
- 操作jQuery集合。
- 迭代jQuery集合元素。

本章将介绍如何使用高度灵活的jQuery()函数来创建新的DOM元素。使用jQuery创建新元素的操作练习会非常频繁。你会发现,使用jQuery方法。实现Ajax时,如何把外部JSON和XML数据注入页面中会非常方便。

此外,还会学习与jQuery()不同的方法。我们会把这些方法分为两部分。首先,将描述方法,包括jQuery集合、接受一个选择器作为参数来创建新的元素集合。例如,将看到一个方法怎样从一个集合开始到创建新的集合,然后让这个新集合包含之前初始化集合的所有元素,设置参数的选择器是可选的,以便起过滤作用。然后介绍与选择器关联性不大的其他方法,但是它们允许我们来迭代集合中的元素,然后测试这些元素。现在就开始学习吧。

52

3.1 创建新HTML元素

Generating new HTML

许多情况下,想要生成新的HTML代码插入网页中。这种动态元素可能是简单的文本,也可能是把复杂的数据库数据显示为一个表格。一种典型的情况是当需要从外部获取数据时,使用Ajax后,返回的数据是JSON或者XML格式。

使用jQuery创建动态元素是相当容易的。可以通过传递给\$()函数一个包含HTML标签的字符串来创建一个jQuery对象。参考下面的代码:

```
$('<div>Hello</div>');
```

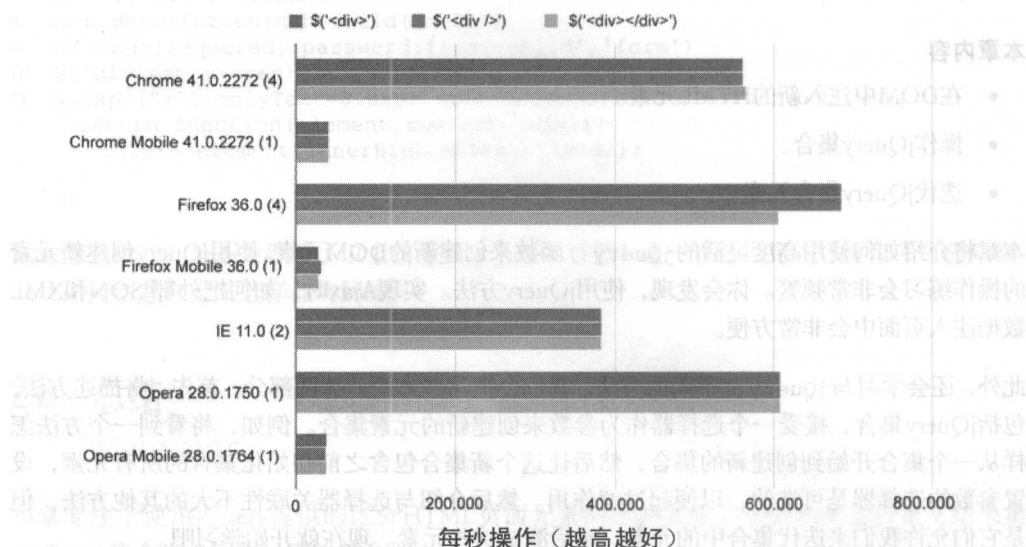
这个表达式创建了一个包含div元素的jQuery对象,该对象可以添加到页面中(此时还没有注入DOM中)。任何在已经存在的元素上执行的jQuery方法都可以在新创建的元素上使用。现在看起来这个机制可能无足轻重,但是当要抛出事件处理器、Ajax及特效时(下面章节里会

看到)，就会发现它十分强大。

注意，如果想创建一个空的div元素，则可以直接使用下面的简写代码：

```
$('<div>');
```

这个语句等价于`$('<div></div>')`和`$('<div/>')`。对于创建包含其他元素的语句，虽然我们强烈推荐使用规范的格式以及包含开始和结束元素的标签，但是从性能的角度来看，以上三种语句都是等价的，正如在图3.1中的测试结果一样。



53 图 3.1 使用三种 jQuery() 方法创建新元素的性能对比，从性能角度来看，在不同浏览器里是一样的

创建这种简单的HTML元素十分简单，在创建更复杂的元素时使用jQuery的链式功能也不会太难。可以在任意的jQuery集合中使用任意的jQuery方法来创建新的元素，也可以使用jQuery的attr()方法来为新元素添加属性（会在后面的章节介绍）。而且jQuery还提供了更好的方法来满足这些需求。

第2章介绍了\$()函数的第二个参数context。当使用\$()函数来创建新元素时，可以使用context参数来为正要创建的新元素指定属性及其值。虽然这些元素还处于JavaScript对象的状态，但是这些对象的属性可以通过元素的属性来设置。

假设要创建一个img元素，它包含多个属性，而且可以点击该图片。看列表3.1中的代码。

列表 3.1 动态创建完整功能的图片元素

```
$( '<img>',  
{  
  src: 'images/little.bear.png',  
  alt: 'Little Bear',  
  title: 'I woof in your general direction',  
  click: function() {  
    alert( $(this).attr('title') );  
  }  
})  
appendTo('body');
```

① 创建基本的 img 元素

② 赋值各种特性

③ 通过 body 元素尾部追加元素来把元素添加到 DOM

建立 click 处理器

列表中的单行jQuery代码创建了基本的img元素①；使用第二个参数设置重要的属性，比如图片地址、文本及悬浮的标题②；并且将其附加到DOM元素上（body元素的子元素）③。这个例子中，使用jQuery的appendTo()方法把元素添加到DOM树中。虽然还没有介绍这种方法，但是它把对象元素追加到jQuery集合中。此时，只有新创建的图片——参数指定的追加目标对象，就是body元素。

这里也做了一个小特效。这个例子中使用了第二个参数来建立事件处理效果，图片被点击时会弹出一个警告框（内容是从图片title属性中抽取的）④。

无论如何组织这些代码，它都非常强大，虽然只有几行代码，且容易阅读，但是包罗万象。这种语句在jQuery支持的页面里并不常见。如果感觉头晕，也不要担心，后面的章节会介绍这种语句里使用的每个方法。编写这种组合式语句也是jQuery的常见编程格式。

图3.2展示了这些代码的效果，图片第一次加载时如图3.2(a)所示，该图片被点击时如图3.2(b)所示。这个例子的完整代码可以在本书的chapter-3/listing-3.1.html中找到。

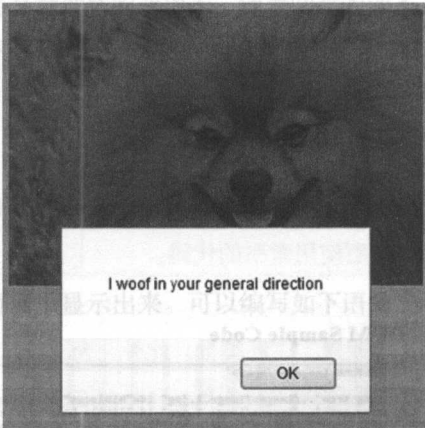


图 3.2(a) 动态创建复杂的元素也相当简单（包含这个图片，点击时会弹出警告框）

图 3.2(b) 动态创建的图片可以使用任何样式和属性，也包括鼠标单击弹出对话框的行为

到目前为止，已经在匹配元素的集合上使用了jQuery处理方法，但是很多时候，希望在对象

执行处理之前执行更复杂的操作。

3.2 管理 jQuery 集合

Managing the jQuery collection

一旦有了jQuery集合，无论这些集合是从现有的DOM元素使用选择器查询而来，还是使用HTML代码块创建而成，接下来都要使用强大的jQuery集合方法来操作这些元素。

我们将在第4章中开始介绍这些方法——如果你想要进一步细化jQuery集合。本节会介绍多种细化、扩展及过滤jQuery结果集的方法。



为了帮你理解这些知识，我们为本章提供了另外一个试验页面：jQuery操作试验页面 (chapter-3/lab.operations.html)。这个页面看起来有点像第2章的选择器试验页面，如图3.3所示。

jQuery Operations Lab Page

Operation








Type any jQuery expression that results in a jQuery set into the text field below and click the Execute button.

Operation:

0 matching element(s):

DOM Sample

Some images:



This is a <div> with an id of someDiv.

Hello, I'm a <h2> element

I'm a paragraph, nice to meet you.

- jQuery website
 - CSS1
 - CSS2
 - CSS3
 - Basic XPath
- jQuery also supports
 - Custom selectors
 - Form selectors

Language	Type	Invented
Java	Static	1995
Ruby	Dynamic	1993
Smalltalk	Dynamic	1972
C++	Static	1983

Text:

Radio group: ☐ A ☐ B ☐ C

Checkboxes: ☐ 1 ☐ 2 ☒ 3 ☐ 4

DOM Sample Code

```
<span>Some images:</span>
<div>
  <img src='../images/image.1.jpg' id='hibiscus' alt='Hibiscus'>
  <img src='../images/image.2.jpg' id='little-bear' title='A dog named Little Bear'>
  <img src='../images/image.3.jpg' id='verbenas' alt='Verbenas'>
  <img src='../images/image.4.jpg' id='cozmo' title='A puppy named Cozmo'>
  <img src='../images/image.5.jpg' id='tiger-lily' alt='Tiger Lily'>
  <img src='../images/image.6.jpg' id='coffee-pot'>
</div>
```

图 3.3 jQuery 操作试验页面，允许实时创建 jQuery 集合，也可以看到如何创建和管理 jQuery 集合

新的页面不仅像选择器试验页面，还可以使用相似的操作方式。在这个页面里，不仅可以输入选择器，还可以输入任何返回jQuery结果集的代码。这个操作在DOM例子的上下文里执行，而且和使用选择器试验页面一样，也会显示结果。

注意：这个试验页面加载的元素取决于在iframe内部操作哪个对象。因为受某些浏览器安全设置的限制，这个操作可能失败。为了避免这个问题，可以在像Apache、Tomcat或IIS这种Web服务器里执行，也可以找一些别的替代方案。例如，基于WebKit内核的浏览器，你也可以通过使用--allow-file-access-from-files，以命令行接口(CLI)的方式来运行。非常重要的一点是，这个命令会创建一个新的进程，所以它不是打开一个选项卡，而是打开一个新的窗口。

56

jQuery操作试验页面允许输入任意表达式，然后返回jQuery结果集。由于jQuery的链式调用机制，这个表达式可以包含jQuery方法，也可以通过动手试验来检查jQuery操作。

当然要输入有效的语法才行，这样才能返回正确的jQuery结果集的表达式。否则，就会出现许多意想不到的JavaScript错误。



为了进一步理解知识，可以在浏览器里打开页面，并在操作框里输入下面的代码。

```
$('img').hide();
```

然后点击“执行”按钮。这个操作在DOM例子的上下文中执行，而且会看到图片从例子中消失。

在任意操作之后，如果想恢复之前的页面，则点击“Restore”按钮就会重新恢复之前的页面。目前还没有详细讲解hide()方法，其实它也属于jQuery中的方法。本书后面会详细介绍。现在，需要知道的就是这个方法可以隐藏元素。我们使用它的目的就是想通过一个具体的例子，让你了解可以在操作试验页面进行何种操作。你将会看到新的选项卡，随着阅读的深入，后面会看到这个页面非常有用，可以测试不同的jQuery操作。

3.2.1 确定集合大小

之前提到过，jQuery集合的行为与数组很像。比如提供了length属性，这与JavaScript类似，指的是jQuery集合中元素的个数。

假设你知道页面中所有的段落数量，现在需要在页面上显示出来。可以编写如下语句显示数量：

```
alert($('p').length);
```

OK，现在已经知道你有多少个元素了。但是如何直接访问它们呢？

3.2.2 从集合获取元素

一旦有了jQuery集合，就可以通过jQuery方法来执行一些排序操作。有时经常需要直接访问元素或者执行原生的JavaScript的操作。下面看一些jQuery操作的例子。

通过索引查找元素

因为jQuery允许把jQuery集合当作JavaScript数组来处理，所以可以使用简单的数组索引来获取集合中的元素。例如，获取带有alt属性的图片元素集合中的第一个元素的代码如下：

```
57 var imgElement = $('img[alt]')[0];
```

细心的读者可能会注意到，没有把美元符(\$)放在变量名之前(img元素)。这并不是我们忘记了，而是因为jQuery集合包含DOM元素的数组，所以如果查找单个元素，它不只是由一个元素组成的jQuery集合，而是一个原始的DOM元素。

如果想选择使用方法而不是索引，那么jQuery也定义了get()方法来支持这个操作。

get()方法语法

get([index])

获取一个或者所有集合中匹配的元素时，如果不指定参数，那么jQuery对象中的所有元素都会作为JavaScript数组返回。如果提供索引参数index，就会返回对应的元素。index可以为负数，此时会从匹配元素集合的末尾开始计算。

参数

index(数值) 要返回单个元素的索引。如果忽略，则所有集合的元素就会作为数组返回。如果指定负数索引，就会从集合尾部开始计算。如果超出范围，就会返回未定义(undefind)错误。

返回

返回结果为一个DOM元素，或者DOM元素数组，或者未定义(undefind)错误。

以下的代码

```
var imgElement = $('img[alt]').get(0);
```

等价于前面使用索引的代码。

get()方法同样可以接受负数索引。使用get(-1)会查找倒数第一个元素，get(-2)会查找倒数第二个元素，以此类推。此外，为了获取单一元素，不提供参数时，get()方法也可以以数组形式返回全部集合元素。

有时候想获取一个包含特定元素的jQuery对象，而不是原始元素。像这样的代码看起来有点奇怪，但是可以正常工作：

```
$$('p').get(2))
```


为此，jQuery提供了eq()方法。它模仿第2章中讨论的选择过滤器:eq()。为了了解两者的区别，下面来看一些代码。假设想选择页面中所有<div>元素集合中的第二个元素，则用下面的代码可一步一步执行筛选操作：

```
var $secondDiv = $('div').eq(1);
```

↑ 使用 eq()方法选择

```
var $secondDiv = $('div:eq(1)');
```

↑ 使用:eq()过滤器选择

58

二者语法的差别很小，但是出于性能考虑（第15章会详细讲解），建议还是选择第一种方法（eq()）。为了获得更高的性能，也强烈建议你使用方法而不是过滤器。

既然我们已经强调过了方法和过滤器之间的差别，那么现在是应该深入学习这个方法的时候了。

eq()方法语法

eq(index)

获取集合中指定索引的元素，返回值是包含结果的新集合。

参数

index (数值) 要返回单个元素的索引。如果指定负数，就会从集合结尾处开始计算。

返回

包含一个或者多个元素的jQuery集合。

获取集合中的第一个元素是常见的需求，有一种更方便的方法是使用:first()。

first()方法语法

first()

获取集合中的第一个元素，返回包含元素的新的集合。如果初始集合为空，则返回空集合。

参数

无。

返回

包含一个或者零个元素的jQuery集合。

first()方法也有对应的选择过滤器:first。再来看一下两种实现方式的对比。若想加入查找页面中所有的段落元素，则可以编写如下代码：

```
var $firstPar = $('p').first();
```

↑ 使用 first()方法选择元素

```
var $firstPar = $('p:first');
```

↑ 使用:first 过滤器选择

不出意外，代码语法上的差别不大，但是first()方法比:first过滤器更好。

59

也许你也会想到，应该有种方法可以获取集合的最后一个元素，这种方法就是选择过滤器：`last`。

last()方法语法

last()

获取集合中的最后一个元素，返回只有该元素的集合。如果初始集合为空，则返回空。

参数

无。

返回

包含一个或者零个元素的jQuery集合。



如果想练习这种方法，则可以使用jQuery操作试验页面。例如，查找页面列表中的第一个元素，可以编码如下：

```
$('#li', '.my-list').first();
```

现在看一下获取集合中数组元素的方法。

作为数组获取所有元素

如果希望获取jQuery对象中的所有元素，并且作为DOM元素数组返回，那么就用jQuery提供的对应方法`toArray()`。

toArray()方法语法

toArray()

作为数组返回DOM元素的集合。

参数

无。

返回

以JavaScript数组形式返回集合中的DOM元素。

思考下面的例子：

```
var allLabeledButtons = $('label + button').toArray();
```

这条语句会查询所有页面上的以`<label>`为前缀的`<button>`元素，作为JavaScript对象转换为数组赋值给`allLabeledButtons`变量。

获取元素的索引

`get()`方法查找的是给定索引的元素，你可以使用反向操作`index()`来确定元素的索引值。

index() 方法语法

index([element])

查找集合中的特定元素，返回集合中的索引值，或者查找其兄弟元素集合的第一个元素的索引。如果没有找到元素，则返回-1。

参数

element (Selector|Element|jQuery) 包含选择器的字符串、元素引用，或者可以确定值的jQuery对象。

如果已经给定了jQuery对象，就会查找集合的第一个元素。如果没有设置参数，就会返回其兄弟所在集合的第一个元素。

返回

集合或者其兄弟元素中指定元素的索引值，或者没有找到，返回-1。

为了帮助大家理解这种方法，我们来看下面的HTML代码例子：

```
<ul id="main-menu">
  <li id="home-link"><a href="/">Homepage</a></li>
  <li id="projects-link"><a href="/projects">Projects</a></li>
  <li id="blog-link"><a href="/blog">Blog</a></li>
  <li id="about-link"><a href="/about">About</a></li>
</ul>
```

由于某种原因，我们想获取包含博客链接ID为blog-link的列表元素()的索引，这些元素位于ID为main-menu的元素内部。

注意：最好不要在页面中充满 ID，因为对于大型的网站应用来说难以管理，难以保持不出现重复 ID。这里只是为了演示才这么操作的。

可以使用下面的语句来获取这个值：

```
var index = $('#main-menu > li').index($('#blog-link'));
```

根据对index() 参数的了解，也可以重构如下代码：

```
var index = $('#main-menu > li').index(document.getElementById('blog-link'));
```

记住索引从0开始计算。第一个元素索引为0，第二个元素索引为1，以此类推。因此，要获取的索引值为2，因为它是第三个元素。这个例子的代码位置为chapter-3/jquery.index.html，也可以在JS Bin(<http://jsbin.com/notice/edit?html,js,console>)中获取。

index() 方法也可以用来查找父节点中元素的索引（兄弟节点）。这种情况可能难以理解，我们来看一个详细的例子。ID为blog-link的元素父节点是ID为main-menu的元素。这些兄弟元素从DOM树结构的角度来看共享一个父节点（无序列表）。相对于标签，这些元素都是其他列表项目。这些链接被排除，因为它们在main-menu内，但是没有和blog-link处于同一级别，不是兄弟关系。

编写代码如下：

```
var index = $('#blog-link').index();
```

index变量的值一样是2。

弄明白无参调用index()方法非常有意思。思考下面的代码：

```
<div id="container">
```

```
<p>This is a text</p>
```

```

```

```
<a href="/">Homepage</a>
```

```

```

```
<p>Yet another text</p>
```

```
</div>
```

这次，标签里包含几个不同的元素。假设想知道父节点(<div>的ID为container)里第一个img元素的索引，可以编写如下代码：

```
var index = $('#container > img').index();
```

index的值为1，因为在container的子元素中，第一个找到元素的是第二个元素(<p>元素之后)。

除了可以查找元素的索引，根据jQuery集合元素的DOM树关系，jQuery也可以支持获取集合的子集。下面来看看如何实现。

3.2.3 使用关系获取集合

jQuery允许根据DOM元素之间的层级关系从现有集合来获取新的集合。

假设有一个段落元素的ID为description，我们想获取它上级<div>元素的个数。基于目前的选择器知识，这是不可能的。这也就是为什么会有parents()这样的专门方法。思考下面的代码：

```
var count = $('#description').parents('div').length;
```

使用parents()，可以查找期望的信息。这个方法可以在当前匹配的集合中查找每个元素的上级元素（包含ID为description的段落元素P）。也可以使用选择器来过滤上级元素，正如例子所示。因为在jQuery集合中只有一个元素（假设页面中存在），所以结果是我们所期望的。

如果想要知道这个假设的段落元素的子元素数量呢？则使用选择器可以轻易实现：

```
62 var count = $('#description > *').length;
```

等一下！这里是不是在使用前面章节里极力不推荐的通用选择器？很不幸，是的。从性能的角度来看，更好的方法是使用children()方法，代码如下：

```
var count = $('#description').children().length;
```

用这种方法并不会返回文本节点。那怎么处理这种情况呢？

对于这种必须处理文本节点的情况，可以使用`contents()`方法。这个方法和`children()`的不同之处在于：前者不接受任何参数。回到计算元素数目的例子上，可以这样编写代码：

```
var count = $('#description').contents().length;
```

你知道，只计算元素个数没有什么用，可能已经迫不及待要使用jQuery来实现一些极佳的效果了。希望你能再耐心学习以下几个页面的内容，以便巩固知识。

`find()`方法可能是最经常使用的方法之一。假设你要搜索遍历集合中所有元素的子节点（深度优先算法），然后返回一个jQuery对象。那么这个新的jQuery对象包含所有匹配选择器表达式的所有元素。例如，假设所有匹配的元素存储在变量`$set`中，就可以获取集合中所有(<p>)段落的引文元素(<cite>)。

```
$set.find('p cite');
```

与许多其他的jQuery方法一样，`find()`的强大之处体现在jQuery的链式调用中。这个方法在jQuery链式调用中需要约束子元素查询的时候非常方便。

在列出所有此类别中的方法之前，来看另外一个例子。有如下HTML代码：

```
<ul>
  <li class="awesome">First</li>
  <li>Second</li>
  <li class="useless">Third</li>
  <li class="good">Fourth</li>
  <li class="brilliant amazing">Fifth</li>
</ul>
```

假设想查询样式为`awesome`的元素的兄弟元素，但是排除样式为`brilliant`和`amazing`的元素。为此，可以使用`nextUntil()`方法。它接受一个选择器参数，查询集合中所有的兄弟元素直到到达匹配的选择器元素位置。因此，可以编写如下代码：

```
var $listItems = $('.awesome').nextUntil('.brilliant.amazing');
```

◀ 63

如果只想查找样式为`good`的元素，则如何做？这个函数的第二个参数是可选参数`filter`，允许你来实现这个目标。你可以修改之前的代码，新的代码如下：

```
var $listItems = $('.awesome').nextUntil('.brilliant.amazing', '.good');
```

也可以在试验页面 chapter-3/jquery.nextuntil.html 中执行，或者访问JS Bin (<http://jsbin.com/fuhen/edit?html,js,console>)。

表3.1展示了这些内容，以及本类别中其他允许你从现有对象获取jQuery新对象的方法。大部分方法都接受可选参数，大部分都是使用方括号包装。

表 3.1 基于 HTML DOM 元素关系获取新集合的方法

方 法	描 述
<code>children([selector])</code>	返回集合中元素的所有子元素，过滤选择器的参数可选
<code>closest(selector[,context])</code>	返回包含匹配指定选择器的每个元素最近上级节点的集合，从元素开始作为第一个参数，元素或者 jQuery 对象都可以作为参数传递。此时，它将根据上级节点进行测试。如果找到，则返回包含此元素的集合，否则返回空数组
<code>contents()</code>	DOM 元素作为可选参数指定给 context（上下文）。此时，匹配的上级节点必须是此元素的子元素
<code>find(selector)</code>	返回集合中每个元素的子节点作为一个集合，根据给定的选择器、jQuery 对象或者元素进行过滤
<code>next([selector])</code>	返回匹配元素集合中的直接兄弟节点，结果以集合形式返回。如果某个元素是多个元素的兄弟，则只返回一次。如果提供了选择器，则只查询匹配选择器的元素的兄弟
64 <code>nextAll([selector])</code>	返回集合中元素的所有兄弟节点，以集合形式返回。如果提供选择器，则它只查询匹配选择器的元素
<code>nextUntil([selector[,filter]])</code>	返回指定元素的兄弟元素，但是不包括匹配选择器的元素。如果没有匹配选择器的元素，或忽略选择器，就会选择所有的兄弟节点元素。此时 selector 可以是包含选择器表达式的字符串、DOM 节点或 jQuery 对象
<code>offsetParent()</code>	这个方法可以接受另外一个可选选择器表达式、过滤器作为第二个参数。如果提供，就会根据过滤器来判断是否匹配进行过滤
<code>parent([selector])</code>	返回集合中元素最近的、相对的、绝对的或者固定位置（CSS 语义定义）的父节点的集合
<code>parents([selector])</code>	返回集合中所有元素的直接父节点元素，结果以集合形式返回。如果元素的父节点多于一个元素，则只会返回一个。如果设置选择器，则只会选择匹配选择器的元素
<code>parentsUntil([selector[,filter]])</code>	以集合形式返回集合中所有元素的唯一祖先（元素只被选择一次，即使匹配多次）。结果既包含直接父节点也包含其他高级节点，但是不包含文档根节点。如果设置选择器，则只会返回匹配的祖先节点
<code>prev([selector])</code>	以集合形式返回集合中所有元素的祖先节点，但是不包括匹配选择器的元素。如果不匹配或者没有提供选择器，则会选中所有的祖先节点。这个例子中，selector 可以是包含选择器表达式的字符串、一个 DOM 节点或者 jQuery 对象
	这个方法可以接受另外一个可选选择器表达式、过滤器作为第二个参数。如果设置，则会测试元素是否匹配元素

方 法	描 述
<code>prevAll([selector])</code>	以集合形式返回集合中所有元素的前置兄弟节点。可以使用选择器过滤
<code>prevUntil([selector], [filter])</code>	以集合形式返回集合中所有元素的前兄弟元素, 但是不包含选择器匹配的元素。如果不匹配选择器或者没有提供选择器, 则选择所有的前置兄弟节点。此时 <code>selector</code> 可以是包含选择器表达式的字符串、DOM 节点或 jQuery 对象 这个方法可以接受另外一个可选选择器表达式、过滤器作为第二个参数。如果设置, 则会测试元素是否匹配元素
<code>siblings([selector])</code>	以集合形式返回集合中元素的兄弟节点。元素可以被选择器过滤

既然已经介绍了所有的方法, 现在就来看看一些具体的例子。

思考一种情况: 一个位于事件处理器中的可以通过 `this` 访问的按钮元素触发事件处理器 (第6章详细讲述)。这种情况下点击按钮经常会执行一些 JavaScript 代码 (例如, 计算或者 Ajax 调用)。

另外, 假设还想找到按钮所在的最近的 `<div>` 元素。那么用 `closest()` 方法可以轻而易举地实现:

```
$(this).closest('div');
```

但是这只会找到最直接的父级元素 `<div>`。假如要找的元素在更高层级的树节点上呢? 则可以重新定义选择器代码来区分这个元素。代码如下:

```
$(this).closest('div.my-container');
```

现在第一个使用样式 `my-container` 的上级 `<div>` 元素就会被选中。

提示一下, 这些方法工作的方式类似。例如, 要选择一个具有特殊 `title` 属性值的兄弟按钮, 代码如下:

```
$(this).siblings('button[title="Close"]');
```

这里进行的操作就是查找所有具备 “Close” 的 `<button>` 标签的兄弟节点。如果只想查找第一个兄弟元素, 则可以使用本章中学习的 `first()` 方法:

```
$(this).siblings('button[title="Close"]').first();
```

这些方法给了我们最大的自由, 可以根据 DOM 元素之间的关系来查找元素。但是如何调整 jQuery 集合中的元素呢?

3.2.4 分割集合

一旦有了集合, 那么或许想通过添加或者删除来调整集合的元素。jQuery 提供了大量的方法

来进行集合的管理操作。首先来看下添加集合元素的操作。

在集合中添加新元素

很多时候需要向现有集合中添加元素，特别在对初始集合使用某些操作方法后特别需要再添加一些元素。记住，jQuery调用链允许在单行语句中执行大量的操作。

我们来看一些具体的例子。首先从简单的场景开始。假设想要查找页面上所有具备alt或者title属性的图片元素

```
66 > $('img[alt],img[title]');
```

为了演示add()方法，也可以使用下面代码来匹配相同的集合：

```
$('img[alt]').add('img[title]');
```

使用add()方法允许我们链式调用多个选择器，以创建满足多个选择器的元素集合。

像add()这样的方法，对于jQuery链式调用来说意义重大（比聚合选择器更加灵活），因为它们不需要最初的集合作为参数，而是创建新的集合。我们会在后面看到它如何与其他的方法如end()（第3.2.5节会详细介绍）联合使用来对原始集合进行一些回退操作。

add()方法的语法说明如下。

add()方法语法

add(selector[,context])

创建jQuery对象，并且添加selector指定的元素到集合中。selector选择器参数可以是包含选择器的字符串、HTML片段、DOM元素、DOM元素数组或者jQuery对象。

参数

selector (Selector|Element|Array|jQuery)指定了想要添加的元素。这个参数可以是选择器，任何匹配的元素都会添加到集合中。如果参数是HTML片段，就可创建适当的元素然后加到集合中。如果是DOM元素，则直接添加。如果是DOM元素数组或者jQuery对象，也会直接添加到集合中。

context (Selector|Element|jQuery)指定了查询匹配元素的范围。这个参数也是传递给jQuery()函数的参数。

返回

包含附加元素的原始集的副本。

在浏览器中打开jQuery试验页面，然后输入下面的表达式：

```
$('td').add('th');
```



点击“执行（Execute）”按钮，就会执行jQuery操作选择所要的表格单元。图3.4展示了结果集的截屏。

在图3.4中，可以看到所有的表格单元，也包含头部标签(<th>)。这些新增到集合里的元素，黑色下画线和灰色背景高亮的元素是被选中的元素。它会为每个元素添加一个名为found-element的CSS样式。

现在来看看更具体的add()方法使用的例子。假设想把所有具备alt属性的图片元素的边框变成红色，CSS样式名字设置为red-border。所有具备alt或者title的图片元素都设置了一个opaque样式。CSS选择器的逗号(,)在这里不起作用，因为你想对一个集合应用操作，然后在另外一个操作之前添加更多新的元素到集合中。虽然使用多条语句可以轻易实现这个需求，但是使用jQuery链式调用语法会更高效、更优雅。添加引用的CSS样式，可以使用名为addClass()的方法。简写形式下，它接收一个样式名作为参数，然后把样式添加给集合中的元素。

jQuery Operations Lab Page

Operation

Type any jQuery expression that results in a jQuery set into the text field below and click the Execute button.

Operation:

Execute

Restore

15 matching element(s):

TH

TH

TH

TD

TD

TD

TD

TD

TD

TD

TD

TD

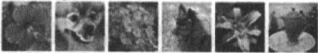
TD

TD

TD

DOM Sample

Some images:



This is a <div> with an id of someDiv

Hello, I'm a <h2> element

I'm a paragraph, nice to meet you.

- jQuery website
 - CSS1
 - CSS2
 - CSS3
 - Basic XPath
- jQuery also supports
 - Custom selectors
 - Form selectors

Language	Type	Invented
Java	Static	1995
Ruby	Dynamic	1993
Smalltalk	Dynamic	1972
C++	Static	1983

Text:

Radio group:

☐ A ☒ B ☐ C

Checkboxes:

☐ 1 ☒ 2 ☒ 3 ☐ 4

Submit

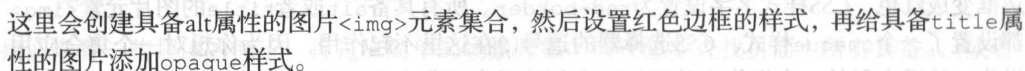
DOM Sample Code

```
<span>Some images:</span>
<div>
  <img src='../images/image.1.jpg' id='hibiscus' alt='Hibiscus'>
  <img src='../images/image.2.jpg' id='little-bear' title='A dog named Little Bear'>
  <img src='../images/image.3.jpg' id='verbena' alt='Verbena'>
  <img src='../images/image.4.jpg' id='cozmo' title='A puppy named Cozmo'>
  <img src='../images/image.5.jpg' id='tiger-lily' alt='Tiger Lily'>
  <img src='../images/image.6.jpg' id='coffee-pot'>
</div>
```

图 3.4 jQuery 表达式匹配的表格单元

用一条语句实现此目标的写法如下：

```
$('img[alt]')  
  .addClass('red-border')  
  .add('img[title]')  
  .addClass('opaque');
```

这里会创建具备alt属性的图片元素集合，然后设置红色边框的样式，再给具备title属性的图片添加opaque样式。



在jQuery试验页面里输入语句，点击“执行（Execute）”按钮，则结果如图3.5所示。

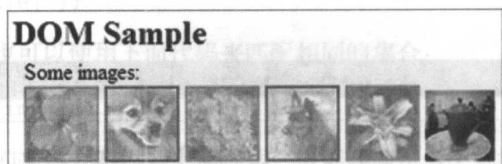


图 3.5 jQuery 链式调用允许在单条语句里执行复杂的操作

结果是看到红色边框的图片（具备alt属性）。

此外，所有的图片除了那个咖啡壶图片，由于应用不透明度样式褪了色，咖啡壶图片是唯一一个既没有alt属性又没有title属性的元素。正如所注意到的，不是花的图片（除了咖啡壶图片）也有黑色边框。原因是，除了opaque样式，之前的found-element样式也被自动添加了。实际上，found-element样式已经添加到所有的图片样式了，但是黑框的鲜花图片已经被red-border样式覆盖了。

add() 方法也可以用来向现有的集合添加元素。传递元素或者数组的直接引用作为add() 方法的参数。如果有一个元素存储在变量someElement中，则可以添加到具备alt属性的图片集合中，代码如下：

```
$('img[alt]').add(someElement);
```

可能这还不够灵活，add() 方法不仅允许我们添加现有元素到集合中，而且可以直接传递HTML标签字符串作为参数来实现这个目标。思考下面的代码：

```
$('p').add('<div>Hi there!</div>');
```


此行代码会创建文档里所有p元素的集合，然后给予这个集合再创建一个包含<div>的新集合。

使用add() 方法扩大集合非常方便，其功能非常强大。现在来看看如何使用jQuery方法来删除集合中的元素。

删除集合元素

我们看到，使用add() 方法把多个jQuery选择器的对象组合到一起相当简单。当然，也可以使

用`not()`方法链式调用选择器来剔除（except）某些元素。这与第2章我们讨论的`:not`选择器相似，它也可以用在jQuery链式调用方法中来删除元素。

假设要选择页面中的所有包含`title`属性但是值不是“puppy”的图片元素，那么可以使用单个选择器来限制条件（`img[title]:not([title*="puppy"])`），为了说明问题，假设已经忘记了`:not`过滤器。现在使用`not()`方法来剔除匹配选择器的元素，表示一种排除关系。为了执行这个操作，可以编写如下代码：

```
$('#img[title]').not('[title*="puppy"]');
```



在jQuery试验页面里输入这个表达式，然后执行。可以看到只有第一个小狗的图片设置成了高亮。那个原始集合中的黑色小狗（black puppy）图片因为包含`title`属性，已被`not()`方法删除，因为它的值是“puppy”。

not()方法语法

not(selector)

创建不包含选择器`selector`参数匹配元素的新集合。

参数

`selector` (Selector|Element|Array|jQuery|Function)指定什么元素会被删除。如果参数是jQuery选择器，就会删除匹配的元素。如果是元素引用、数组或者jQuery集合，那么这些元素都会从集合中删除。如果传输的是函数，那么会针对每个函数来执行函数，然后删除元素并返回`true`。此外，jQuery也可以传递内部元素的索引作为函数的第一个参数，而且当前元素作为第二个参数。

返回

删除元素后原始集合的新副本。

`not()`方法可以传递单个元素或者数组的引用来删除元素。后者非常有趣，而且强大，正如我们会记住的，任何jQuery的集合度都可以作为元素数组的引用。

当需要最大的灵活性时，可以传递函数给`not()`方法，然后决定是不是挨个来删除元素。jQuery传递给函数一个指定集合中元素的索引参数。思考下面的例子：

```
$('#div').not(function(index) {  
    return $(this).children().length > 2 && index % 2 === 0;  
});
```

这段代码会选择页面所有的`<div>`元素，然后删除集合中包含两个以上子元素的`div`以及奇数索引（不是位置）的`div`。如果要查看结果，则可以尝试在试验页面进行测试，就可以看到四个匹配的结果。

这个方法允许通过编程方式过滤集合元素，处理复杂的或者无法使用单个选择器表达式表示的情况。

当传递给`not()`函数进行测试时，我们期望不符合条件的结果，`not()`函数与`filter()`方法

相反。not()函数的工作方式类似，只是当函数返回false时会删除元素。

假设想要创建所有包含正整数的<td>元素的集合。这种情况下，可以使用filter()方法，代码如下：

```
$('#td').filter(function() {  
    return this.innerHTML.match(/^d+$/);  
});
```

此语句会创建一个td元素的集合，然后调用传递给filter()方法的函数来对每个匹配的当前元素调用操作。这个函数使用正则表达式来确定元素内容是否匹配（一个或者多个数字），如果不匹配，就返回null。元素过滤器返回null，通常false(null、undefined等)并不包含在集合中。

filter()方法的语法说明如下。

filter()方法语法

filter(selector)

创建一个集合副本，然后从新集合中删除不符合selector选择器条件的元素。

参数

selector (Selector|Element|Array|jQuery|Function) 指定要删除的元素。如果参数包含选择器的字符串，那么任意不匹配的元素都会被删除。如果传递的是元素引用、元素数组或jQuery对象，那么所有这些元素都会从集合中删除。如果传递的是函数，则该函数将会调用集合中的每个元素（this引用的当前元素），并返回false导致元素从集合中删除。此外，jQuery传递元素索引作为函数的第一个参数，当前元素作为第二个参数。

返回

71 返回一个集合的副本，不包含被删除的元素。



再次打开jQuery试验页面，输入之前的表达式，然后执行。我们会看到只有Invented列的表格单元的td元素被选中。

filter()方法也可以用作选择器表达式。这样使用正好与not()方法的相反，删除任意不匹配选择器的元素。它并不是超强的方法，它通常用来限制选择器。当然也可以在jQuery链式调用中使用，例如：

```
$('#img')  
    .addClass('opaque')  
    .filter('[title*="dog"]')  
    .addClass('red-border');
```

以上代码会选择页面中的所有图片元素，然后设置opaque样式，再过滤只有title属性为dog的元素，最后设置red-border样式。结果就是所有的图片都是半透明的，但只有棕褐色的狗的图片元素获得了红色边框。

`not()` 方法和 `filter()` 方法是调整集合元素的强大工具, 可以基于任意不同的条件进行操作。也可以基于集合中元素的位置获取集合的子集。下面来看看这些方法。

获取集合的子集

有时, 我们希望基于现有集合中元素的位置来获取子集。jQuery 提供了 `slice()` 方法来实现这个功能。此方法会返回原始集合的任意连续的片段集合。

slice()方法语法

slice(start[,end])

创建并返回匹配集合中部分元素的新集合。

参数

start(Number) 首元素位置为0, 包含在返回的片段集合中。

end(Number) 返回元素的最后位置, 索引从0开始计算。如果为负数, 则表示偏移从末尾开始。如果不提供参数, 则片段会选择到集合末尾结束。

返回

新创建的集合。

如果想根据初始集合中的某个元素来获取新集合, 则可以使用 `slice()` 方法。例如, 为了获取之前选择的第三个元素, 可以使用如下代码:

```
$('#img, div.wrapper', 'div').slice(2, 3);
```

这条语句会选择所有的 `` 图片元素, 以及包含 `wrapper` 样式的 `<div>` 元素, 然后在 `div` 元素内部生成一个只有第三个元素的新集合。正如我们看到的, 之前学习的知识又出现在眼前。

注意: 这与使用 `get(2)` 不同, 它会返回集合中的第三个 DOM 元素, 但是与使用 `eq(2)` 的效果一样。

如语句

```
$('*').slice(0, 4);
```

用于选择页面中的元素, 然后创建包含前四个元素的集合。

为了获得直到集合结尾的元素, 语句可以这样编写:

```
$('*').slice(4);
```

它会匹配页面上所有的元素, 然后返回除了前四个元素之后的所有元素。

另外一个获取子集的方法就是 `has()`。与 `:has` 过滤器一样, 这个方法会测试所有 jQuery 对象中的子元素, 使用它可以选择元素以包含在子集合中。

has()方法语法

has(selector)

创建并返回新的集合，只包含匹配selector选择器的子元素。

参数

selector(Selector|Element) 包含选择器的字符串会应用到集合中元素的所有子元素上，或者测试DOM元素。

返回

jQuery对象。

例如，思考下面的代码：

```
$('#div').has('img[alt]');
```

此代码会创建包含所有<div>元素的集合，然后创建并返回至少有一个子元素和alt属性的<div>元素的集合。

转换集合的元素

通常需要转换集合的元素。例如，也许你想搜集集合中所有元素的ID或者form元素值来创建查询字符串。map()方法非常适合这种情况。

map()方法语法

map(callback)

在集合中的每个元素上调用callback函数，然后返回值到一个jQuery对象中。

参数

callback(Function) 集合中每个元素执行的回调函数。函数传递两个参数：集合中元素的索引及元素本身。元素被作为函数的上下文（this关键字）。为了向新集合添加元素，就必须返回与null及undefined不同的值。

返回

返回转换过的值。

例如，用下面的代码会搜集页面上所有<div>元素的ID：

```
var $allIDs = $('#div').map(function() {  
    return this.id;  
});
```

使用这条语句，可以查询包含所有ID的jQuery对象，但这个对象可能不是我们想要的。如果想要原始的JavaScript数组，则可以在链式调用toArray()方法时做转换：

```
var allIDs = $('#div').map(function() {  
    return this.id;  
}).toArray();
```

```
})  
.toArray();
```

本章还想介绍其他的方法。现在来看看其他的方法。

遍历集合元素

map()方法对于遍历集合元素后搜集值或者转换为其他元素非常有用,但是很多时候是为了达到其他目的才迭代元素。此时,jQuery的each()方法更是无价的。

each()方法语法

each(iterator)

遍历集合里的所有元素,然后为每个元素调用传入的iterator(迭代器)函数。

参数

iterator(Function) 针对匹配集合中的每个元素都调用迭代函数。迭代函数接受两个参数:集合中元素的索引及元素本身。元素被作为函数的上下文(this关键字)。

返回

jQuery集合。

74

这种典型的例子就是为匹配集合中的每个元素设置一个属性值。例如,

```
$('img').each(function(i){  
    this.alt = 'This is image[' + i + '] with an id of ' + this.id;  
});
```

这段代码会对页面上的每个img图片元素执行传入的函数代码,使用元素的ID和索引来修改元素的alt属性。

目前为止,已经看了许多使用jQuery对象的方法,但是还没有完呢!现在再来看看其他处理jQuery对象的方法。

3.2.5 使用集合的其他方法

jQuery还有几个技巧可让我们完善对象的集合。

有一个方法就是,检查一个集合是否包含至少一个元素来满足给定的选择器表达式。大于等于1个元素匹配选择器时,is()方法返回true;否则返回false。看下面的例子:

```
var hasImage = $('*').is('img');
```

如果当前页面至少包含一个图片元素,则这条语句返回的结果会将hasImage值设置为true。

is(selector)

确定集合中是否有元素匹配给定的选择器。

参数

selector(Selector|Element|Array|jQuery|Function) 用选择器表达式、元素、数组元素或者jQuery对象来测试集合中的元素。如果提供了函数，就为集合中的每个元素调用这个函数（this设置到每个项目）。调用时返回true会导致整个函数返回true。此外，jQuery也可以传递集合中元素的索引作为函数的第一个参数，当前元素作为第二个参数。

返回

如果至少有一个元素匹配给定的选择器，就返回true；否则返回false。

在jQuery中，它是高度优化的快速方法，而且推荐使用在关注性能的情况下。已经了解了在单个语句中执行jQuery链式调用的强大特性，现在继续学习使用它，因为这个特性非常重要。链式调用可以在单个简化的语句中实现强大的操作，是非常高效的方式，因为集合不需要每次都重新计算后再执行不同的方法。

现在来考虑下面的语句：

```
$('img').filter('[title]').hide();
```

这条语句会产生两个集合：包含DOM中所有元素的集合及包含title属性的元素的集合（是的，可以使用单个选择器实现这项操作，这里是为了演示说明的需要。想象一下，假设要在调用filter()方法之前执行某些重要的操作）。

隐藏集合中的所有元素（包含title属性的元素）后，假如要随后调用某些方法，怎么办？比如在过滤集合元素后添加新的样式名称该怎么办？不能在链式调用后再执行，这样会影响带标题的图片，而不是最初的图片集合。

这时就需要jQuery提供end()方法。当使用在jQuery调用链中时，用这个方法就会回退到之前的jQuery集合，然后后续的操作就可以用到之前的集合上。

例如，下面的语句：

```
$('img')
  .filter('[title]')
  .hide()
  .end()
  .addClass('my-class');
```

其中filter()方法会返回包含title属性的图片集合。通过调用end()方法，可以回退到之前的集合中（最初的集合元素），然后调用addClass()方法来设置样式。不需要干预end()方法，addClass()方法只会操作带有title属性的图片集合。为了避免这个问题，可以把最初的集合存储到一个变量里，然后编写两条语句。end()方法允许我们不定额外变量而在单条语

句里执行所有的操作。

`end()` 方法的语法说明如下。

`end()` 方法语法

`end()`

在jQuery链式调用中使用该方法；在当前调用链中结束最近的过滤操作，然后返回匹配元素的集合到之前的状态。

参数

无。

返回

上一个jQuery集合。

jQuery对象维护了一个内部栈来保存针对匹配元素集合的修改。当调用诸如之前介绍的方法时，新的集合会被压栈。一旦调用jQuery的`end()`方法，顶部的集合就会弹出，留出前一个集合来操作后续方法。

76

jQuery提供的另外一个方便的方法就是`addBack()`，它会把栈上前一个集合的元素添加到当前集合里，可以选择性提供选择器参数。

`addBack()` 方法语法

`addBack([selector])`

把栈上前一个集合的元素添加到当前集合里，可以选择性提供选择器参数。

参数

`selector(Selector)` 包含选择器的字符串，用来匹配当前集合的元素。

返回

合并的jQuery集合。

思考下面的代码：

```
$('div')
  .addClass('my-class')
  .find('img')
  .addClass('red-border')
  .addBack()
  .addClass('opaque');
```

这段代码会选择页面上的所有`<div>`元素，为每个元素添加`my-class`样式，再创建一个包含所有`div`的子图片`img`元素的新集合；然后设置`red-border`红色边框样式，并创建第三个集合，这个集合是`div`元素（堆栈顶部的元素集合）和子图片`img`元素的合集；最后，它又为元素设置了`opaque`样式。

最后的结果就是，<div>元素集合设置了my-class和opaque样式，而这些元素的子图片元素都被设置了redborder和opaque样式。

本章证明了精通选择器和调用方法是十分重要的。无论需求多么复杂，这些特性经常联合起来就可遍历DOM元素，允许我们精确地选择元素。既然已经牢牢掌握了这些知识，现在来学习更激动人心的技术主题。

3.3 总结

Summary

本章介绍了如何创建和查找匹配的元素集合，以及如何使用HTML代码动态创建新的元素集合。这些独立的元素存储在集合中，最后附加到页面文档的某个部分上。为了调整集合，jQuery提供了许多方法，既有创建后立即执行的，也有链式调用方法中执行的。

对于现有的集合，使用过滤条件也是创建新的jQuery集合的快捷方法。

总之，jQuery提供了许多工具以确保可以轻易、准确区分页面上要操作的元素。

本章也介绍了许多与DOM元素操作无关的扩展性问题。但是，既然已经知道如何选择我们想要的操作元素集合，现在就应该来学习实现页面动态效果的jQuery DOM操作方法了。

使用特性、属性和数据

Working with properties, attributes, and data

本章内容

- 获取和设置元素属性。
- 使用元素特性。
- 在元素上存储自定义数据。

每个软件开发者都应该有一个非常重要的认识：即使是大型复杂的软件，也包含基本的指令组合。求和、计数项及遍历元素仅是这些基本操作的几个例子。同样，可以通过jQuery操纵属性、特性、类、样式等创建一幅很好的画面。

可以使用JavaScript的本机函数操纵属性和特性的元素，但有些任务并不像我们想的那样容易完成。此外，使用这些功能可留下处理浏览器不兼容的负担。jQuery提供一套完整的方法来轻松处理属性和特性，以帮我们解决所有兼容性问题。

当我们在处理DOM元素和特效创建时的另外一个关键概念，就是排序存储在元素上的自定义数据的可能性。jQuery允许我们保存元素在某一时刻的状态。这是创建插件的关键，在本书的第三部分会介绍。

本章重点介绍很多jQuery提供的用于处理属性、特性和数据的方法。

4.1 定义元素特性和属性

Defining element properties and attributes

当涉及DOM元素时，一些最基本的组件，可以将操作的特性和属性分配给这些元素。这些特性和属性最初被分配到JavaScript对象实例，表示DOM元素由于解析HTML标记，在脚本控制下可以动态更改。应确保我们明白这些概念和术语。

特性是固有的JavaScript对象，每个特性都有一个名字和一个值。在脚本控制下，JavaScript的动态特性允许我们创建JavaScript对象的特性（如果是刚刚接触JavaScript的新手，附录对这个

概念有详细的讲解)。

属性指的是在标记中指定的值DOM元素，不是对象实例的特性。考虑下面的HTML标记的图

```

```

这个元素的标记中，标记的名称是img，标记的id、src、alt、class和title表示元素的属性，每一个属性由名称和值组成。在DOM中，浏览器通过读取并解释此元素标记来创建组成的JavaScript HTMLElement类型对象实例。

这两个概念的第一个区别是特性的值可能不同于其相关属性的值。而后者始终是字符串，相对应的特性值可以是字符串、布尔值、数字甚至对象。例如，给你一个字符串，将尝试检索的tabIndex作为HTML属性（由所有的数字组成，且仍然是一个字符串）。

检索其相关的特性会给出一些数字。另一个例子是样式，如果作为特性检索，则其值为对象(CSSStyleDeclaration类型)；而作为属性检索，则其值为一个字符串。在操作中要看到这种差异，下面看看网页下面的HTML元素：

```
<input id="surname" tabindex="1" style="color:red; margin:2px;" />
```

现在相同页面创建包含以下状态的脚本，或者直接在浏览器控制台下键入这些语句：

```
var element = document.getElementById('surname');
console.log(typeof element.getAttribute('tabindex')); // 打印"string"
console.log(typeof element.tabIndex); // 打印"number"
console.log(element.getAttribute('style')); // 打印"color:red; margin:2px;"
console.log(element.style); // 打印 CSSStyleDeclaration 对象包含的所有样式
```

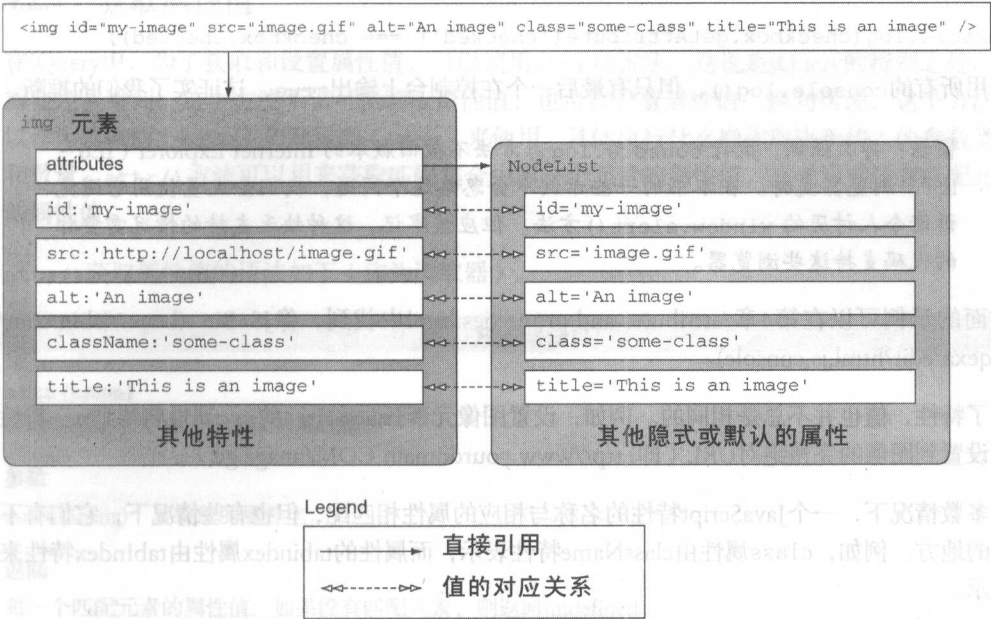
元素的所有属性封装成一个对象，它们作为命名特性，有足够理由相信，属性等存储在DOM元素实例上。另外，对象表示的元素被赋予大量的特性，包括一些表示该元素标签的属性。

因此，属性值不仅存储在属性中，还存储在少量的其他特性中。图4.1展示了这一过程的简要概述。

活动链接仍然存在于属性对象中存储属性值和相应的特性之间。更改属性值的结果是更改相应的特性值，反之亦然。具体来说，考虑下面输入的元素(一个复选框，要精确)增加了另一个非标准属性(以粗体显示)：

```
<input type="checkbox" id="book" name="book" title="Check this!"
      book="jQuery in Action" />
```

HTML标记



```
console.log(checkbox.getAttribute('title') === checkbox.title);  
console.log(checkbox.getAttribute('checked') === checkbox.checked);
```

调用所有的`console.log()`，但只有最后一个在控制台上输出`true`，这证实了我们的推断。

警告：再次强调，执行`console.log()`方法不被旧版本的 Internet Explorer (IE6 ~ IE7) 浏览器支持。在本书的一些示例中会忽略这个问题，我们会大量使用它避免诉诸令人讨厌的`window.alert()`方法。但应该牢记，这种缺乏支持的情况需要你的代码支持这些浏览器。

前面的示例可以在第4章/attributes.and.properties.html中找到，像JS Bin (<http://jsbin.com/soqexa/edit?html,js,console>)。

除了特性，值也并不是总是相同的。例如，设置图像元素`image.gif`的`src`属性将导致`src`特性被设置到图像的完整绝对URL（即`http://www.yourdomain.COM/image.gif`）。

大多数情况下，一个JavaScript特性的名称与相应的属性相匹配，但也有些情况下，它们有不同的地方。例如，`class`属性由`className`特性表示，而属性的`tabindex`属性由`tabIndex`特性来表示。

对属性支持的检测

HTML5引入了几个新的属性，可以将其添加到页面元素中。在标记中定义属性之间的区别，当需要检测这些属性的支持时，不存在由浏览器产生的DOM元素就能起作用。比如新的HTML5需要的属性。当用户将其添加到一个表单字段时，浏览器需要用户输入一些数据到该字段后才能提交。可以使用下面的代码检测浏览器是否支持此属性：

```
if ("required" in document.createElement("input")) {  
    // 支持属性  
}
```

如果浏览器支持功能，则此测试将返回`true`；否则返回`false`。

如果需要检测支持更多的功能，那么建议你使用Modernizr，这是第1章中提到的库。

jQuery能让你轻松地操纵元素的属性和特性的方法，并提供访问元素的实例，这样也可以更改其特性。选择哪一项操作，取决于我们想做什么。下面先来看看如何获取和设置元素的属性。

4.2 使用属性

Working with attributes

本节会深入属性的世界，学习jQuery提供的处理属性的方法。

4.2.1 获取属性值

在jQuery中，为了获取和设置属性值，可以使用`attr()`方法。这也是jQuery的特别之处。以后会经常看到，这个方法不仅可以读取属性值，也可以设置属性值。换句话说，这个方法可以作为读取器（getter）和设置器（setter）来使用。具体执行什么操作取决于传入的参数类型和数量。`attr()`方法可以用来获取匹配集合中第一个元素的属性值，或者设置所有匹配元素的属性值。

`attr()` 获取属性值的语法如下（作为读取器）。

`attr()`方法语法(1)

`attr(name)`

获取匹配集合中第一个元素的属性值。

参数

`name(String)` 要获取的属性名。

返回

第一个匹配元素的属性值。如果没有匹配元素，则返回`undefined`。

◀ 83

通常，虽然元素的属性都是HTML预定义的，某些时候也可以使用`attr()`方法来设置自定义属性。正如第2章中看到的，可以使用HTML5中的`data-*`来添加自定义属性。为了演示这个过程，我们来看下面的为图片元素添加自定义属性（粗体高亮）的例子：

```

```

注意，已经为元素添加了自定义属性`data-custom`。也可以使用下面的语法查询属性值：

```
$('#my-image').attr('data-custom');
```

HTML中的属性名字对大小写不敏感，所以不会在意诸如`title`属性是大写还是小写：`Title`、`TITLE`和`TiTLlE`这些都是一样的效果，都可以正常访问。在XHTML中，标签中的属性名必须小写，但是也可以使用大小写名称来查询。

给`attr()`方法设置不同值就有不同的特征。我们来具体看看。

4.2.2 设置属性值

jQuery中有两种方法可以设置属性值。先来看最简单的方法。这种方法允许我们一次设置所有元素的单个属性值，语法如下。

attr(name, value)

为jQuery对象中的所有元素设置命名的属性和值。

参数**name(String)** 属性的名字。**value(String|Number|Boolean|Function)** 指定属性的值。它可以是任意的JavaScript表达式。除非设置函数, 否则任何其他值都会转换为字符串。集合中的每个元素都会调用函数、传递元素的索引和当前命名属性的值。函数的返回值变成属性值。**返回**

jQuery集合。

attr()的新用法看起来简单, 实际相当复杂。

在最基本的形式中, value参数可以是任意的JavaScript表达式, 只要能最终转化为字符串就行。如果value参数是一行函数代码或函数的引用, 则会更加有趣。此时, 会针对每个元素来调用函数, 将函数的返回值作为属性值。当调用函数时, 它会传递两个参数: 一个是从0开始的集合中元素的索引, 另外一个当前属性的值。此外, 元素会作为函数调用的上下文(this), 允许函数针对每个元素做特殊的工作——这是使用函数方式的强大之处。

思考下面的代码:

```

$('[title]').attr('title', function(index, previousValue) {
    return previousValue + ' I am element ' + index +
        ' and my name is ' + (this.id || 'unset');
});

```

这个方法将会返回页面上包含title属性的所有元素。它会修改每个元素的title属性值, 使用id、元素索引和其他字符串组合起来。如果id不存在, 就使用'unset'字符代替。

当需要初始值来计算新的属性值时, 可以使用这种方式来设置属性的值, 无论属性值是否依赖元素的其他数据, 或者也可以单独设置元素的属性。

attr()方法的另外一个重要的特征就是它允许一次指定多个属性值。

attr()方法语法(3)

attr(attributes)

使用传入属性和值来设置匹配集合所有元素的对应属性。

参数**attributes(Object)** 对象的值会复制给集合中所有元素的属性。**返回**

jQuery集合。

此方法是一次性为集合中所有元素设置多个属性的快捷方式。传递的参数可以是任意对象引用、对象常量，对象的特性名和值用来设置属性。思考下面的代码：

```
$('input').attr({
  value: '',
  title: 'Please enter a value'
});
```

这段代码会把所有input元素的value设置为空字符串，然后把title设置为'Please enter a value'(请输入一个值)。

注意，如果作为value参数的传递对象的任意特性值是个函数引用，则它的操作方式与attr()类似，函数会为jQuery对象中的每个元素调用一次。

警告：通过document.createElement()方法来修改input或者button的type属性时，IE6~IE8浏览器上会抛出异常。

现在已经知道如何获取和设置属性的值，那怎么删除这些属性呢？

4.2.3 删除属性

jQuery提供了removeAttr()方法用于删除DOM元素的属性值，其语法说明如下。

removeAttr()方法语法

removeAttr(name)

删除指定的属性，或者删除所有匹配集合元素的属性。

参数

name(String) 要删除的属性名称或者名称的集合，用空格分割。

返回

jQuery集合。

现在来看例子中使用的这个方法。我们的目标是要删除网页中所有图片的title和alt属性。为了达到目的，编写如下代码：

```
$('img').removeAttr('title alt');
```

removeAttr()方法内部使用了JavaScript的removeAttribute()函数，它的优势就是可以在jQuery对象中的每个元素上调用并支持链式调用。

删除属性并不会删除JavaScript DOM元素对应的特性，虽然可能会带来属性值的变化。例如，删除readonly属性将会导致元素的readonly属性值从true变为false，但是其特性本身并没有从元素上删除。

现在看看如何把这些知识用到实际的页面中。

4.2.4 玩转属性

86 让我们看看如何用这些方法处理各种不同的页面元素属性。

例子#1: 强制在新窗口中打开超链接

假设要在新窗口中打开一个外部网站的链接。这非常简单, 可以直接使用`target`属性即可, 代码如下:

```
<a href="http://external.com" target="_blank">Some External Site</a>
```

这个解决办法不错, 但是, 如果没有办法控制HTML标签元素呢? 可能允许的是内容管理系统或者wiki, 用户可以自己添加内容, 而我们没有办法来为每个链接添加`target="_blank"`。首先要确定你想要什么: 让所有具备`href`属性并且以`http://`开头的链接元素可以通过把`target`属性设置为`_blank`来实现能在新窗口中打开。此外, 不想修改那些已经设置了正确值的元素。为了简化起见, 例子中故意忽略了其他的协议, 比如FTP和HTTPS。

可以使用本节学习的知识来简化实现以下代码:

```
$( 'a[href^="http://"]' )  
  .not( '[target="_blank"]' )  
  .attr( 'target', '_blank' );
```

首先要选择页面上带有`href`属性并且值是以`http://`开头的超链接标签元素, 它表示要链接到外部网(假设内部网站使用的不是绝对地址, 而是相对地址)。然后可以把所有超链接元素的`target`属性值为`_blank`的元素排除。最后可以把所有其他的超链接元素的`target`属性值设置为`_blank`。这些操作只需要简单的一行jQuery代码就可以实现了。可以在chapter-4/new.window.links.html例子页面中查看原始的例子代码。

例子#2: 模拟 placeholder 属性

另外一个使用jQuery属性功能的绝佳情况就是模拟新的HTML5的`placeholder`属性。在这个例子中, 我们展示了一个级别的模拟实现例子, 虽可以显示`attr()`方法的使用, 但并不是要在生产环境下使用。首先, 不会测试浏览器是否支持, 所以这个代码也可以在支持`placeholder`属性的浏览器中执行操作。其次, 一旦字段被聚焦, 文本就无法被隐藏, 所以用户只能手动删除属性值。

placeholder 属性

`placeholder`属性显示文本, 它指定要显示的字段或者字段内部可能值的例子。文本会一直显示直到字段聚焦或者用户输入了值, 这取决于浏览器, 此时, 文本被隐藏。这个属性会应用到`<input>`和`<textarea>`元标签上。在旧浏览器(IE6~IE9)上不支持这个属性会被忽略, 不会执行任何方法, 就好像从来没有写过这种代码。一个可能值的例子在一个字段内。

87

这个例子中我们要做的工作就是复制placeholder属性值，然后把它作为value属性的值设置。这样，无论浏览器是否支持，每个浏览器都会显示占位符的值。因为例子的原因，我们将只介绍

假设已经有了如下的表单代码：

```
<form>
  <label for="username">Username:</label>
  <input id="username" name="username" placeholder="JohnDoe" />
  <label for="email">Email:</label>
  <input type="email" id="email" name="email" placeholder="email@fake.com" />
  <input type="submit" value="Login" />
</form>
```

要满足这些需求，可以编写如下的简单代码：

```
$( 'input' ).each(function(index, element) {
  var $element = $(element);
  $element.attr('value', $element.attr('placeholder'));
});
```

选择所有的元素并遍历它们

创建新的 jQuery 对象并存储到变量里

复制占位符的值，作为 value 特性的值

执行代码的结果如图 4.2 所示。此外，可以在文件 chapter-4/placeholder.html 和 JS Bin (<http://jsbin.com/onuMiDU/edit?html,js,output>) 中找到这个例子代码。

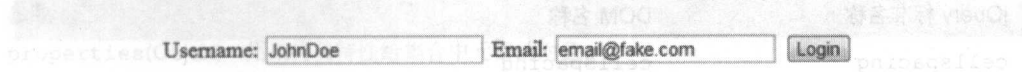


图 4.2 使用 attr() 方法把表单字段的值设置为 placeholder 属性值

现在已经看了如何获取和设置属性值，下面再看如何操作元素的属性。

4.3 操作元素特性

Manipulating element properties

在jQuery中可使用attr()方法来获取和设置特性值，也可以使用相似的prop()方法实现。attr()方法和prop()方法在功能和参数方面十分相似。此外，对于那些喜欢追根问底的人来说，prop()方法其实就是取自attr()方法，且只关注于属性本身。在1.6版本之前，attr()会处理属性和特性。当然这非常久远了。

prop()方法获取特性值的语法介绍如下。

prop(name)

获取匹配集合中第一个元素给定的属性值。

参数

name(String) 要查询的属性的名字。

返回

匹配元素的特性值。如果没有元素或者没有值，就返回undefined（未定义）。

再思考checkbox（勾选框）元素的代码：

```
<input type="checkbox" id="legal-age" name="legal-age" title="Check this!" />
```

可以使用prop()方法来检验它是否被选中。在这个例子中，假设完全忘记了:checked过滤器和is()方法。

```
if ($('#legal-age').prop('checked') === false) {
    console.log('The checkbox is currently unchecked');
}
```

jQuery的prop()方法提供了访问常见特性的方式，由于浏览器的依赖性，这些特性已经随处可见。这里列举了attr()方法使用的标准属性名称，如表4.1所示。

表4.1 jQuery prop() 标准化访问的名称

jQuery 标准名称	DOM 名称
cellspacing	cellSpacing
cellpadding	cellPadding
class	className
colspan	colSpan
contenteditable	contentEditable
for	htmlFor
frameborder	frameBorder
maxlength	maxLength
readonly	readOnly
rowspan	rowSpan
tabindex	tabIndex
usemap	useMap

勾选框（checkbox）选中或者没有选中都可以，但是怎么通过编程方式来选中它呢？可以作为设置器使用prop()方法，其语法如下。

prop()方法语法(1)

prop(name, value)

为jQuery集合中的所有元素设置给定命名的属性和值。

参数

name(String) 要设置的特性的名称。

value(Any|Function) 自定特性的值。它可以是任意的JavaScript表达式或者函数。函数会针对每个元素调用，传递jQuery集合中元素的索引以及命名的属性的当前值。函数返回值会变为特性的值。

返回

jQuery集合。

也可以通过编程方式来设置勾选框的值，代码如下：

```
$('#legal-age').prop('checked', true);
```

正如我们指出的，attr()方法和prop()方法有很多共同之处，一次指定多个特性的扩展功能，也不算例外。prop()方法的语法详细描述如下。

prop()方法语法(2)

prop(properties)

通过给定的对象来为匹配集合中每个元素设置特定的特性和值。

参数

properties(Object) 用来拷贝贝特性给集合中元素的对象。

返回

jQuery集合。

这种格式允许一次设置多个特性，以避免prop()过长的链式调用语句。例如，可以编写如下代码：

```
$('#input:checkbox').prop({  
  disabled: true,  
  checked: true  
});
```

最后一个要讨论的特性管理方法是removeProp()。它可以通过使用prop()来删除选中元素的属性，其语法介绍如下。

removeProp()方法语法

removeProp(name)

删除jQuery集合中每个元素的属性。

参数

name(String) 要删除的特性名称。

返回

jQuery集合。

与removeAttr()不同,这个方法不支持以空格分割的名称列表。这个方法不应该用来删除原生特性,比如checked或者required,因为它会完整地删除特性。一旦删除,就无法再次添加到元素上。如果想要修改某个元素的特性,则可以使用prop()方法将值设置为false即可。

对于数据来说,HTML和W3C中定义的元素特性是非常有用的概念,但是在网页制作的过程中,经常需要存储自定义的数据格式。下面看看jQuery如何处理这些问题。

4.4 元素中存储自定义数据

Storing custom data on elements

让我们直截了当地说出来:全局变量太烂了。除了不常用的全局变量,你很难想象还有更糟糕的用来存储复杂页面行为需要同时定义和实施网页的地方。它不仅存在作用域问题,而且当同时促发多个操作时还难以扩展伸缩(打开和关闭菜单、激活Ajax请求、执行动画,等等)。

JavaScript的功能本性可以通过闭包帮助解决这种问题(如果需要重新复习本知识点,可以阅读附录部分),但是闭包并不适用于所有情况。

因为页面行为是面向元素的,所以可以使用元素本身来存储数据。由于JavaScript的天然属性,因此可以动态地为对象创建自定义属性。但是必须小心,因为DOM元素是由JavaScript对象来代表的,所以与其他对象一样,可以使用自定义特性来扩展这些对象。

这些自定义特性,称为expandos,并非没有风险。尤其是很容易创建循环引用,会导致严重的内存泄露,例如,保留针对某个不需要元素的引用。在传统的Web程序里,DOM元素会被频繁丢弃,加载新的页面时,内存泄露可能还不是最严重的问题。但是,当开发高交互性的Web程序,尤其是使用了大量脚本的页面时,可能加载需要很长时间,此时的内存泄露可能是一个巨大的问题。

jQuery以可控、不依赖于潜在的有问题的expandos自定义属性方式提供了存储数据到任意DOM元素的方法。可以让任意的JavaScript值,甚至数组和对象,使用data()方法存储到DOM元素上。这个语法的说明如下。

data()方法语法(1)

data(name, value)

使用jQuery方法为集合中的所有元素添加传递的数据。

参数

name(String) 要存储的数据的名称。

value(Any) 要存储的值，除了undefined(未定义)。

返回

jQuery集合。

data()方法不会区分驼峰命名法的变量名字的大小写，与虚线和横线链接的名字一样对待，下面的代码

```
$('.class').data('lastValue');
```

等价于：

```
$('.class').data('last-value');
```

此外，与attr()方法通常用于保存字符串不同，data()可以保存数据的类型。当作为读取器时，data()也可以尝试把属性值转换为原生的数据类型。我们来看下面的例子。

假设页面中有这样的代码：

```
$('.class').attr('last-value', 10);
console.log(typeof $('.class').attr('last-value'));
```

← 打印"string"

那么可以在控制台上看到"字符串"，因为attr()方法已经把10转换为与字符串等价的("10")。换句话说，如果写的是

```
$('.class').data('last-value', 10);
console.log(typeof $('.class').data('last-value'));
```

← 打印"number"

则，在控制台看到的是“数字”，因为data()方法保留的值是数据类型的。

下面思考HTML代码：

```
<input id="name" name="name" data-mandatory="true" />
```

使用attr()方法和data()方法会获取不同的值，代码如下：

```
console.log(typeof $('#name').attr('data-mandatory'));
console.log(typeof $('#name').data('mandatory'));
```

← 打印"string" ← 打印"boolean"

一方面，使用attr()方法查询的值可作为字符串，所以查询的类型是"string"。另一方面，使用data()方法会把值转换为Boolean（同样应用到number、null等），所以看到的是"boolean"。

undefined并不会作为一个值对待，但是仍然会返回一个jQuery对象。因此，如下的语句

```
$('#name').data('mandatory', undefined);
```

不会修改mandatory的值，但是它会返回一个jQuery对象，这个对象允许进行链式调用。

能为元素添加新数据，这功能非常不错，但是目前还是受限于一只能添加一个项目，这不太实用。幸运的是，jQuery提供data()方法变量，作为setter器，可以接受键-值（key-value）对对象作为参数。

data()方法语法(2)

data(object)

为集合中的每个元素添加键-值对对象数据。

参数

object(Object) 存储键-值对数据的对象。

返回

jQuery集合。

继续探索data()方法的特性之前，我们要提醒一下，jQuery也为jQuery对象提供了工具方法，与之前介绍的data()方法的使用方式类似。

jQuery.data() (或者等价的\$.data())属于低级别的方法，因为它是一个DOM元素操作，而不是jQuery对象。这个方法接受的参数与data()的一样，但是还引入了一个新参数（参数列表中的第一个），传递要存储数据的元素。

为了便于理解它们的不同，我们假设有一个元素，ID为book，但我们想使用\$.data()（此方法对jQuery对象无效）方法来存储数据，就可以使用下面的代码：

```
$.data(document.getElementById('book'), 'price', 10);
```

如果使用data()（这个对于jQuery对象有效），可以这样编写代码：

```
$('#book').data('price', 10);
```

不要诧异，data()方法不仅可以用于写数据，还可以用于读数据。下面是关于data()方法查询数据的介绍。

data()方法语法(3)

data([name])

使用指定的名称来查询存储的数据或者HTML5 data-*属性。如果没有指定名字，则会返回所有存储的数据。

参数

name(String) 要查询数据的名称。

返回

返回查找的数据，如果没有找到，就返回undefined。

data()方法作为读取器非常有趣，但可能会导致一些混淆，所以下面进一步深入讨论。

此方法的读取器形式使用设置器版本帮助查询存储在内存中的数据。如果没有找到之前存储的数据，则方法会使用给定的名字查找HTML元素的data-*属性。一旦data()找到data-*值，此方法就会把这个值存储在jQuery管理的存储区（把它作为jQuery内部使用的存储数据的区域）。因此，即使使用attr()方法修改了属性值，任意后续的方法调用也不会再从属性中查找数据值，因为这些数据已经存储在jQuery内存中。若没有找到属性，就会返回undefined。图4.3展示了这个过程。

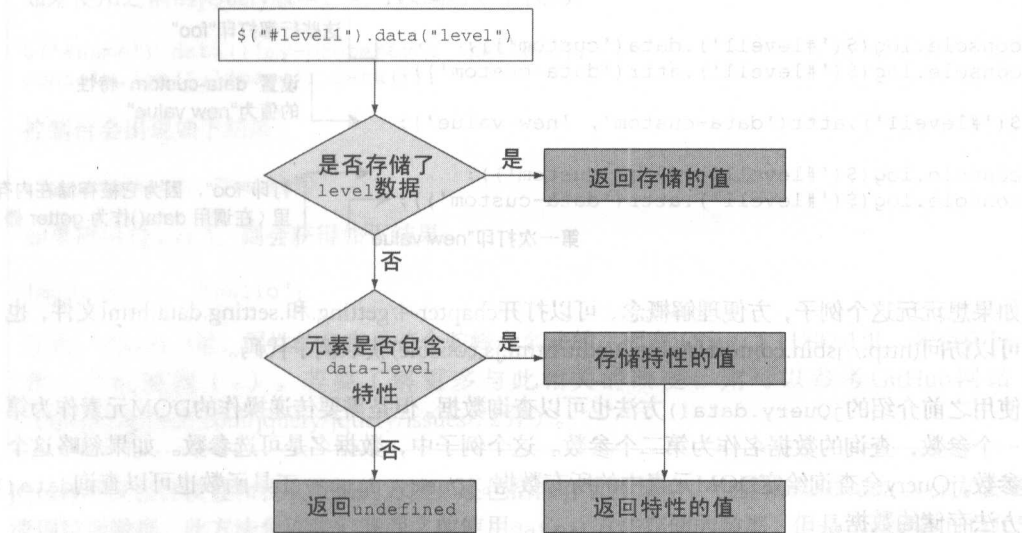


图 4.3 用 data() 方法如何查找存储的数据。此方法的读取器形式使用设置器版本可以查询存储在内存中的数据。如果没有找到必须存储的数据，则方法会使用给定的名字查找 HTML 元素的 data-*特性。如果没有找到这个特性，就会返回 undefined

这个过程有点复杂，所以现在来看一些例子。假设有下面的元素代码：

```
<input id="level1" type="text" value="I'm a text!" data-custom="foo" />
```

若想要查询data-custom的值，则可以使用data()或者attr()方法，但是参数不同。如下所示：

```
console.log($('#level1').data('custom'));
console.log($('#level1').attr('data-custom'));
```

打印两次“foo”，但是方法需要不同参数

前面两条语句都会打印出“foo”结果。如果使用data()方法呢？

```
$('#level1').data('custom', 'new value');
console.log($('#level1').data('custom'));
console.log($('#level1').attr('data-custom'));
```

更新 data 特性的值

使用 data()打印值。字符串是“new value”

使用 attr()打印值。字符串是“foo”

使用data()和attr()方法去查询的时候，data()方法改变了custom的值。这个结果完全不同！听起来很疯狂？还没有完成呢。

正如上一个例子，假设还没有添加任何数据，页面刚加载完毕。这时想要证明：一旦使用data()方法把一个数据从HTML元素中查询出来，那么这个值就会被data()方法来独立管理，因此，后续调用attr()方法也不会影响这个结果。为了实际观察，可以使用data()方法来查询数据，然后使用attr()方法来修改属性值，最后再调用attr()和data()方法来比较不同点。下面是具体的实现代码：

```
console.log($('#level1').data('custom'));
console.log($('#level1').attr('data-custom'));

$('#level1').attr('data-custom', 'new value');

console.log($('#level1').data('custom'));
console.log($('#level1').attr('data-custom'));
```

这些行都打印"foo"

设置 data-custom 特性的值为"new value"

打印"foo"，因为它被存储在内存里（在调用 data()作为 getter 器）

第一次打印"new value"

95 如果想玩玩这个例子，方便理解概念，可以打开chapter-4/getting.和.setting.data.html文件，也可以访问(<http://jsbin.com/uHAzIyoD/edit?html,js,console>)获取例子代码。

使用之前介绍的jQuery.data()方法也可以查询数据。但是需要传递操作的DOM元素作为第一个参数，查询的数据名作为第二个参数。这个例子中，数据名是可选参数。如果忽略这个参数，jQuery会查询给定DOM元素中的所有数据。jQuery.data()工具函数也可以查询data()方法存储的数据。

jQuery 3: 修复 bug

jQuery 3修改了data()方法的一个bug，这个bug在处理数字名称的属性时会出现，例如，如果有如下的元素定义：

```
<div id="name" data-foo-42="test"></div>
```

如果使用jQuery 3之前的版本编写代码，

```
console.log($('#name').data());
```

它不会获取包含属性foo-42为test的对象，而是返回空对象。

jQuery 3: 修改功能

jQuery 3修改了data()方法的行为,以符合Dataset API文档(<http://www.w3.org/TR/html5/dom.html#dom-dataset>)。特别地, jQuery主要的修改是, jQuery会采用驼峰命名法转换所有的属性。为了帮助大家理解这个问题,思考下面的元素代码:

```
<div id="name"></div>
```

如果使用之前的jQuery版本,则可以编写如下代码:

```
$('#name').data({'my-property': 'hello'});  
console.log($('#name').data());
```

控制台会出现如下结果:

```
{my-property: "hello"}
```

如果使用jQuery 3,则会获得如下结果:

```
{myProperty: "hello"}
```

注意: jQuery 3里,属性的名字是符合驼峰命名法的,而在jQuery 1.11和2.1里,名字会包含一个短横线(-)。若要了解更多与此相关的改变,则可以参考GitHub网站(<https://github.com/jquery/jquery/issues/2257>)。

jQuery不仅包含设置和获取数据的方法,还包括内存管理策略,也提供removeData()方法来清理垃圾数据。此方法允许我们清理之前使用data()方法存储的数据,但是不会删除HTML5 data-*属性的值(需要removeAttr()方法)。此方法的语法说明如下。

96

removeData()方法语法

removeData([name])

使用给定的名字删除jQuery对象中所有元素之前存储的数据。参数可以是数组,也可以是以空格分割的名称。如果没有参数,则表示删除所有的值。

参数

name (String/Array) 要删除的数据的属性名,包含名字或者以空格分割的多个名字。如果设置为数组,则它的元素会用来查找要删除数据的名字。

返回

jQuery集合。

基于刚才学习的内容,要删除元素中存储的所有数据,就可以编写如下代码:

```
$('#legal-age').removeData();
```

如果要删除foo和bar数据，则可以编写如下代码：

```
$('#legal-age').removeData(['foo', 'bar']);
```

或者

```
$('#legal-age').removeData('foo bar');
```

注意：当使用jQuery方法删除DOM元素的时候，没有必要手动删除数据。jQuery库非常聪明地处理了这些问题。

对于data()方法，jQuery提供了与removeData()等价的工具函数。这个工具函数称为jQuery.removeData() (或者\$.removeData()，使用jQuery简称)，它接收DOM元素为第一个参数，要删除的数据名为第二个参数。因此，如果想要使用\$.removeData()方法来删除存储在ID为legal-age中的所有数据，则可以编写如下代码：

```
$.removeData(document.getElementById('legal-age'));
```

除了jQuery.removeData()方法外，jQuery还提供了另外一个工具函数来处理元素中存储的数据。这个方法称为jQuery.hasData()，允许我们测试DOM元素是否存储了数据。它的语法介绍如下。

jQuery.hasData()方法语法

jQuery.hasData(element)

确定元素是否包含相关的数据。

参数

element(Element) 要检查的DOM元素。

返回

97 如果包含相关的数据，就返回true；否则返回false。

为了掌握如何使用这个方法，我们来测试ID为legal-age元素中是否包含数据；存储一些数据，然后再进行测试。因此，我们期望的结果是第一次测试返回false，第二次测试返回true。下面实际使用jQuery.hasData()方法：

```
$.hasData(document.getElementById('legal-age')); // 返回 false; 表明此元素没有存储数据
$.data(document.getElementById('legal-age'), 'count', 10); // 存储(10)与名字管理
$.hasData(document.getElementById('legal-age')); // 返回 true, 因为用前面的语句存储了数据
```

在后续章节的高级功能中，会大量使用这种在DOM元素中存储数据的功能。但是，如果遇到过在全局变量中存储数据的头痛问题，那么在元素层次中存储数据的方法带来了完全不同的

体验。本质上，DOM数已经变成可以使用的完整的“命名空间”；存储数据不需要局限在某个单个的空间中。

本章前面已经提到了className属性，作为例子，应介绍标签属性名(markup attribute name)和特性名(property name)的不同。事实上，样式名(class name)在其他方面有些特殊，jQuery的处理方式也比较特殊。第5章会介绍一种比直接使用className属性或者使用attr()方法更好的处理样式名的方法。

4.5 总结

Summary

本章的内容已经超越了选择元素的范畴，开始学习如何操作属性和特性。本章介绍了属性和特性(attribute和property)的不同，以及如何使用jQuery来操作它们。此外，也介绍了如何执行基本的操作，例如，强制在新窗口中打开链接——仅仅是添加或者修改了一个属性。

本章的另外一个重要的知识点是如何管理自定义数据。正如我们看到的，一些方法的行为非常有趣，但是可能导致混淆。希望大家通过例子的学习能够清晰掌握这些知识。

更新或者删除特性、数据以及属性是非常有用的，而且还需要进一步学习jQuery的强大之处。也需要理解概念，比如删除元素、创建包含其他元素的元素，以及处理事件。第5章将深入学习这些主题。

本章主要介绍了如何操作DOM元素，包括如何添加、删除、修改元素，以及如何遍历元素。本章还介绍了如何操作元素的属性、特性、数据以及样式。本章最后介绍了如何操作元素的HTML内容。

本章主要介绍了如何操作DOM元素，包括如何添加、删除、修改元素，以及如何遍历元素。本章还介绍了如何操作元素的属性、特性、数据以及样式。本章最后介绍了如何操作元素的HTML内容。

使用jQuery操作页面

Bringing pages to life with jQuery

本章内容

- 操作元素的样式名。
- 设置DOM元素的内容。
- 获取和设置元素的值。
- 克隆DOM元素。
- 通过添加、移动或者删除来修改DOM树。

今天的Web开发者和设计者比10年前的了解得更多，可通过DOM脚本来增强用户体验，而不是使用闪烁的文字或者动画图片。无论是增量显示的内容，还是通过HTML创建输入控件来提供的基本设定，或者允许用户根据喜好调整网页，DOM操作已经可以让很多Web开发人员带给用户不同的惊喜。

几乎每天，都有许多人上网，打开网页的第一感觉就是“嗨，我不知道你能做出这种功能！”但是从专业开发人员的角度来说不会感到好奇，第一直觉就是查看页面的源代码，看看它是怎么实现的。这就是Web之美，可以查看任何开发人员的代码，你感觉呢？

jQuery提供了强大的工具库来操作DOM元素，而不是需要编写所有的脚本代码，仅用少量的代码就可以做出功能酷炫的网页。第4章介绍了如何使用jQuery来处理属性、特性及数据，本章将讨论如何做出狂赚酷炫的动态页面效果。

5.1 修改元素的样式

Changing element styling

第4章提及了className属性，它就是一个典型的标签属性(attribute)名与特性名不同的例子。但是，坦白来讲，样式名确实有些特别，jQuery的处理方式也如此。本节会介绍一种比使用className属性或者使用jQuery的attr()更好的方法。

当想要修改元素的样式时，有两种方法使用得更频繁。第一种就是通过添加或者删除class属性来导致元素样式重新设置。第二种就是直接为DOM元素添加样式定义。

下面先从简单的jQuery修改元素样式的例子开始，使用的是样式定义方式。

5.1.1 添加与删除样式名

HTML元素的class属性对于创建交互式界面来说至关重要。在HTML中，class属性值通常是字符串，或者空字符分割的字符。虽然可以间隔多个字符，但是通常都是使用一个。例如，下面的代码：

```
<div class="some-class my-class another-class"></div>
```

不幸的是，样式名的值是以空格分割多个值的字符串形式出现，而不是以COM属性数组的形式出现。太失望了！这意味着，每次从已有样式的元素上添加或者删除样式的定义，当读取的时候都要解析每个样式名字符串，以确保在写回样式名的时候以正确的空格分割来保存。

从jQuery或者其他类似的库中得知，HTML5中已经引入了更好的解决办法解决这个问题，即通过一种称为classList（样式列表）的API实现。后者与jQuery提供的标准方法差不多。但不幸的是，与jQuery不同，原生的方法一次只能工作在一个元素上。如果想为集合中的全部元素添加样式，就必须迭代所有的集合。此外，作为新引入的方法，旧的浏览器无法支持，比如IE6~IE9。为了更好地理解这些差别，思考下面用原生JavaScript编写的代码，它会选择所有样式为some-class的元素，然后添加样式hidden；

```
var elements = document.getElementsByClassName('some-class');
for(var i = 0; i < elements.length; i++) {
    elements[i].classList.add('hidden');
}
```

前面的代码只兼容新的浏览器，包括IE10以上。现在来看等价的jQuery代码：

```
$('.some-class').addClass('hidden');
```

这个jQuery版本的代码不仅短小精悍，而且兼容从IE6开始的浏览器（取决于使用的jQuery版本）！

注意：样式名是无须排列的；空格分割的样式名的顺序实际没有任何意义。

虽然编写添加或者删除样式名的代码并不繁重，但是，通常最好的办法就是封装抽象这些底层操作的细节。幸运的是，你不需要开发自己的代码，因为jQuery已经完成了这些工作。

对于前面代码块里为集合中的每个元素添加样式的操作，实际使用addClass()方法是非常容易实现的。

addClass()方法语法

addClass (names)

为集合中的元素添加指定的样式名。如果提供了函数参数，则集合中的每个元素都会传递给函数，一次一个，返回值会作为样式名。

addClass()方法语法

参数

names(String|Function) 指定样式名, 或者空格分割的多个字符串。如果提供函数, 那么该函数会为每个元素调用一次, 那个元素会作为调用函数的上下文(this)。该函数接受两个值: 元素索引及元素的当前样式值。函数的返回值作为新样式名的值。

返回

jQuery集合。

101 使用removeClass()方法删除样式名就更加直截了当。

removeClass()方法语法

removeClass(names)

从jQuery集合中删除每个元素指定的样式。如果提供了函数, 就会为每个元素调用函数, 返回值作为删除样式的名字。

参数

names(String|Function) 指定要删除的样式名, 或者空格分割的多个字符串。如果提供了函数, 就会为每个元素调用函数, 当前元素作为调用函数的上下文(this)。函数接受两个值: 元素索引和要删除的样式名。函数返回值会作为要删除的样式名的值。

返回

jQuery集合。

removeClass()方法非常有用, 假设页面中有如下代码:

```
<p id="text" class="hidden">A brief description</p>
```

可以使用简单的语句来删除隐藏样式:

```
$('#text').removeClass('hidden');
```

很多时候, 我们想要来来回回多次切换属于一个样式(class)名的样式, 可能是在两种状态之前切换, 但是对于界面来说是有意义的。jQuery提供了便捷的实现方法toggleClass()。

toggleClass()方法语法

toggleClass([names][, switch])

为没有样式的元素添加指定的样式名, 或者删除已经存在的样式。注意: 每个样式都会独立测试, 所以有些元素可能已经包含了样式, 有些元素可能删除了样式。

如果设置switch参数, 当switch为true时, 那么样式名通常会添加到没有此样式的元素上。如果switch为false, 就会从元素中删除该样式。

如果采用无参调用方法, 则集合中所有元素的样式都会被删除, 而且重新调用这个方法进行恢复。

如果只设置switch参数，则集合中每个元素的样式名将会根据switch的值保留或删除。

如果提供了函数，则返回值会作为样式名，根据switch的值来执行操作。

参数

names(String|Function) 指定样式名，或者空格分割的样式名。如果是函数，则会针对每个元素来调用函数。当前元素会作为当前函数上下文(this)。该函数接受两个值：元素索引和元素样式的值。函数的返回值会作为切换样式名。

switch(Boolean) 控制表达式，它的值决定是添加(true)还是删除(false)元素的样式。

返回

jQuery集合。

102

正如我们看到的，toggleClass()方法给了很多可能性。在学习其他方法之前，先来看一些例子。

在快速切换元素的可视化效果时，toggleClass()方法特别有用，通常要根据元素来实现。想象要开发一个简单的共享微件，它包含一个按钮，当点击的时候会显示一个盒子，并包含分享到社交媒体的按钮，再次点击按钮时，盒子就会隐藏掉。

使用jQuery及jQuery的click()方法（该方法将在第6章讲解），能轻易实现这个微件小工具：

```
$('.share-widget').click(function() {
    $('.socials', this).toggleClass('hidden');
});
```

这个例子的完整代码在chapter-5/share.widget.html文件中。你失望之前，我们要强调一点，这个页面并没有提供任何实际的分享功能，只有placeholder文本。结果页面的两种状态(盒子隐藏和显示)如图5.1(a)和图5.1(b)所示。

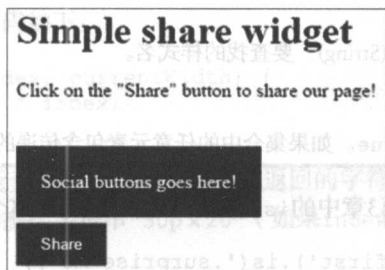


图 5.1(a) 当用户点击按钮的时候，hidden 样式 图 5.1(b) 当点击“Share”按钮时，它会切换为会不停切换，导致盒子显示或者隐藏。hidden 样式。此图展示了盒子展示的状态
初始状态的盒子是隐藏的

基于元素以及拥有的样式进行切换并不是常见的操作，但是，根据其他条件来切换就比较常见。

思考下面的代码：

```
if (aValue === 10) {  
    $('p').addClass('hidden');  
} else {  
    $('p').removeClass('hidden');  
}
```

对于这种情况，jQuery提供了switch参数。我们可以简化上面的代码，即：

```
103 > $('p').toggleClass('hidden', aValue === 10);
```

这个例子中，如果aValue等于10，那么样式hidden会被设置给所有的段落元素；否则会被全部删除。

正如上一个例子，想象要根据给定的条件为每个元素添加样式。例如，要为集合中的每个奇数位置的元素添加样式hidden，而偶数位置的元素保持不变。通过传递函数作为toggleClass()的第一个参数，可以很方便地实现这种效果：

```
$('p').toggleClass(function(index) {  
    return (index % 2 === 0) ? 'hidden' : '';  
});
```

有时需要先检查某个元素包含特定的样式，然后再执行操作。使用jQuery，可以调用hasClass()方法：

```
$('p:first').hasClass('surprise-me');
```

如果元素包含指定的样式，那么这个方法会返回true。此方法的语法介绍如下。

hasClass()方法语法

hasClass(name)

确定集合中的元素是否包含指定的样式。

参数

names(String) 要查找的样式名。

返回

返回true。如果集合中的任意元素包含传递的样式名，就返回true；否则返回false。

回想第3章中的is()方法，也可以实现这个目标。代码如下：

```
$('p:first').is('.surprise-me');
```

但是，hasClass()方法的可读性更强，而且从内部来讲，hasClass()方法更高效。

通过CSS的样式名来操作元素的样式是非常强大的工具。但是，人们有时候想要为元素直接设置样式定义代码。下面看看jQuery如何满足这个需求。

5.1.2 获取和设置样式

修改样式名允许为元素设置任意预定义的样式。但是，有时候只想提前设置一些未知属性的值，然而样式名还不存在。直接在元素上添加样式代码(通过DOM元素的style属性)会自动覆盖预定的样式(有些例外，比如!important，但是这里不会详细介绍CSS规范)，可以对单独元素的样式进行更细粒度的控制。

jQuery css() 方法允许我们操作这些样式，其使用方法与attr()方法的类似。可以通过指定名称和值或者一系列样式来设置单个对象的CSS样式。

css()方法语法(1)

css(name, value) 例如font-size不是20px而是20px，这是因为px是默认单位。

css(properties) 为每个匹配元素命名的CSS样式属性设置指定的值。

参数
name(String) 要设置CSS属性的名字。接受两种CSS和DOM的属性(如background-color与backgroundColor)。大部分情况使用的是第一种格式。

value(String|Number|Function) 字符串、数字或者包含属性值的函数。如果传递数字参数，jQuery会把数字转换为字符串，并在字符串尾加上“px”。如果需要不同的单位，则可以在后面添加自己的单位。如果传递的函数作为参数，则它会为集合中的每个元素调用函数，元素作为函数上下文(this)。该函数接受两个值：元素索引和当前值。返回值作为CSS属性的新值。

properties(Object) 指定一个对象，它的属性会拷贝给集合中的每个元素作为CSS属性。

返回
jQuery集合。

value参数也可以是一个函数，其使用方法与attr()方法的类似。这意味着可以扩展集合中所有元素的宽度，通过20像素乘以元素的索引，代码如下：

```
$('.expandable').css('width', function(index, currentWidth) {  
    return parseInt(currentWidth, 10) + 20 * index;  
});
```

这段代码要传递当前值给parseInt()函数，因为元素的宽度是以像素格式返回的字符串(例如"50px")。如果不做转换，最后的结果就是直接拼接字符串"50px20"(如果index的值是1)。

这个例子中，要通过20像素来扩展每个元素的宽度，jQuery提供了更简单的方法。不需要编写函数，可以直接编写简化代码：

```
$('.expandable').css('width', '+=20');
```

同样，如果想要缩减指定像素的宽度，也可以实现：

105 `$('.expandable').css('width', '-=20');`

另外一个有趣的例子——jQuery如何简化设置元素的透明度（opacity）的属性，设置为0.0~1.0。它可以完美地跨浏览器支持，不会再被IE alpha filter问题困扰了。

现在来看下css()方法第二个签名的使用例子：

```
106 > $('p').css({  
    margin: '1em',  
    color: '#FFFFFF',  
    opacity: 0.8  
});
```

这段代码会为集合中的每个元素设置指定的值。但是，如果想要为元素设置一种逐渐降低的透明效果呢？

正如attr()方法的快捷方式一样，可以使用函数作为任意CSS属性的值，它们会为集合中的每个元素设置调用。可以通过使用函数作为opacity的值，而不是固定的数组来实现这种效果：

```
$('p').css({  
    margin: '1em',  
    color: '#1933FF',  
    opacity: function (index, currentValue) {  
        return 1 - ((index % 10) / 10);  
    }  
});
```

这个例子的代码可以在chapter-5/descending.opacity.html以及JS Bin (<http://jsbin.com/cuhexe/edit?html,js,output>)中找到。

最后来讨论如何使用css()方法：使用样式名或者名字的数组作为参数，查询jQuery对象中匹配名字的第一个元素关联的样式属性。当我们说计算的样式（computed style）时，指的是所有超链接、嵌入的以及内联的CSS代码。

css()方法语法(2)

css (name)

根据指定的name查询集合中首个元素的CSS属性的计算值。

参数

name(String|Array) 指定CSS属性的名称，用来计算返回值。

返回

计算值作为字符串或者属性-值对。

106 > css()方法一直返回字符串类型的结果，所以，如果需要数字或者其他类型，则需要视情况转换类型，比如使用parseInt()或者parseFloat()。

为了明白css()方法传递数组名字的工作原理，下面来看一个例子。目标是在控制台上显示包

含样式special的元素属性及对应的值：font-size、color和text-decoration。为了实现这个目标，编写如下代码：

```
var styles = $('.special').css([
    'font-size', 'color', 'text-decoration'
]);
for(var property in styles) {
    console.log(property + ': ' + styles[property]);
}
```

使用特性值对检索对象

在对象上循环

这些代码可以在chapter-5/css.and.array.html以及JS Bin(<http://jsbin.com/mimixu/edit?html,css,js,console,output>)中找到。在浏览器里加载页面（或者JS Bin），可以看到页面里定义的所有样式的值被打印出来。例如font-size不是20px而是24px，这是因为special样式定义了24px而不是div元素定义的20px。

css()方法也是jQuery解决跨浏览器兼容性的典型例子之一。为了使用原生方法来实现这个目标，需要为Chrome、Firefox、Opera、Safari和从IE9浏览器使用getComputedStyle()方法，而且为IE8之前的版本使用currentStyle和runtimeStyle属性。

继续介绍之前，强调两个重要的事实。首先，不同的浏览器可能返回不同的CSS颜色值，它们逻辑上相等但是文字上不等。例如，如果定义了color:black，则有些浏览器返回#000、#000000或者rgb(0, 0, 0)。其次，对于简写的CSS属性，比如margin或者border，jQuery无法保证（虽然在某些浏览器里支持）。

对于经常访问的一些小的CSS值集合，jQuery提供了便捷的方法来访问它们，然后转换为最常见的使用数据类型。

获取或者设置尺寸

当要设置页面元素的CSS样式时，还有比元素的宽度和高度更常见的属性吗？可能没有，所以jQuery提供了非常便捷的方法来处理数字类型元素的尺寸而不是字符串。

特别是可以通过调用width()和height()方法来方便地使用数值类型来设置元素的宽度和高度。可以按照如下方式设置宽度和高度。

width()和height()方法语法(1)

width(value)

height(value)

设置集合中每个匹配元素的高度和宽度。

参数

value(Number|String|Function) 要设置的值。可以是数值像素或者字符串指定值的单位(比如px、em或者%)。如果没有设置代码，则默认是px（像素）。

width()和height()方法语法(1)

如果提供了函数，就会为集合中的每个元素调用函数，当前元素作为上下文(this)。函数接受两个值：元素索引及当前元素的值。函数的返回值会作为新的值。

返回

jQuery集合。

记住，它们是css()的快捷方式，所以

```
$('#div').width(500);
```

等价于

```
$('#div').css('width', 500);
```

也可以像下面这样来查询宽度和高度。

width()和height()方法语法(2)

width()

height()

查询jQuery对象的第一个元素的宽度和高度。

参数

无

返回

以像素为单位的宽度或者高度。如果jQuery对象为空，则返回null。

这两个方法的获取版本与css()对应的方法不同。css()返回包含值的字符串(例如"40px")，而width()和height()返回数值，它不包含单位，而且直接转换为Number数据类型。如果样式使用了不同于像素px(em、%等)的单位定义宽度和高度，则jQuery仍然会返回元素宽度和高度的像素值。

jQuery 3:bug 修复

jQuery 3修复了width()、height()及其他方法的bug。这些方法不再返回以像素为单位的近似值，这种做法在某些情况下很难确定元素位置。为了理解这个问题，假设有三个元素的宽度为33%，在一个宽度为100像素的容器内部：

```
<div class="wrapper">
  <div>Hello</div>
  <div>Hi</div>
  <div>Bye</div>
</div>
```

jQuery 3之前的版本，如果查询三个子元素的宽度，

```
$('#wrapper div:first').width();
```

就会得到33这个值，因为jQuery取33.33333的近似值。而在jQuery 3中，这个bug已经被修复了，可以获得更加精确的值。

这些函数直接返回数值类型的宽度和高度值，并不是唯一带来的便捷之处。如果曾经尝试通过`style.width`或者`style.height`去查找元素的宽度或高度，那么必须面对的可悲的事实是，这些属性只能通过该元素的相应样式属性设置。通过属性来查找元素的尺寸，必须提前设置它们。这并不是一种典型的做法！

`width()`和`height()`方法用于计算并返回元素的尺寸。在简单的页面中，元素的布局按照自己的尺寸展示。虽然知道元素的精确尺寸并不是必须的，但是对于高交互性的页面来说，知道这种精确尺寸可以正确地放置活动的元素，比如上下文按钮、自定义工具提示、扩展控件及其他动态组件。

我们举个例子说明一下。图5.2展示了一个包含两个主要元素的例子页面：第一个是

元素，包含一个段落文本(加粗的边框和背景颜色)作为测试主题；第二个是

元素，用来展示尺寸。为了在第二个div里写入尺寸，可以使用`html()`方法。后面会简要介绍这种方法。

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam eget enim id neque aliquet porttitor. Suspendisse nisl enim, nonummy ac, nonummy ut, dignissim ac, justo. Aenean imperdiet semper nibh. Vivamus ligula. In in ipsum sed neque vehicula rhoncus. Nam faucibus pharetra nisi. Integer at metus. Suspendisse potenti. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin quis eros at metus pretium elementum.

700x90

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam eget enim id neque aliquet porttitor. Suspendisse nisl enim, nonummy ac, nonummy ut, dignissim ac, justo. Aenean imperdiet semper nibh. Vivamus ligula. In in ipsum sed neque vehicula rhoncus. Nam faucibus pharetra nisi. Integer at metus. Suspendisse potenti. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin quis eros at metus pretium elementum.

338x180

图 5.2 测试元素的高度和宽度不是固定的，取决于浏览器窗口的宽度

测试主题的尺寸事先并不知道，因为没有指定样式规则。元素的宽度是通过浏览器窗口的宽度来确定的，它的高度取决于需要多少空间来展示内部的文本。调整浏览器窗口的大小将会导致宽度和高度的变化。

在页面中，我们使用`width()`和`height()`方法定义一个函数来获取测试主题div的尺寸(id为`test-subject`)，并在第二个div(id为`display`)中显示结果：

```
function displayDimensions() {  
    $('#display').html(  
        $('#test-subject').width() + 'x' + $('#test-subject').height()  
    );  
}
```

我们在脚本的最后调用这个函数，以显示不同的结果，如图5.2所示。

关于浏览器窗口调整事件的处理器，也添加了针对同一个函数的调用(会在第6章里学习)，如图5.2所示。这个页面的完整代码可以在chapter-5/dimensions.html文件里找到。

列表 5.1 动态跟踪和显示元素的尺寸

```
<!DOCTYPE html>
<html>
  <head>
    <title>Dynamic Dimensions Example - jQuery in Action, 3rd edition</title>
    <link rel="stylesheet" href="../css/main.css"/>
    <style>
      #test-subject
      {
        background-color: #FFFFCC;
        border: 2px ridge maroon;
        padding: 0.5em;
      }
    </style>
  </head>
  <body>
    <div id="test-subject">
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
      Aliquam eget enim id neque aliquet porttitor. Suspendisse
      nisl enim, nonummy ac, nonummy ut, dignissim ac, justo.
      Aenean imperdiet semper nibh. Vivamus ligula. In in ipsum
      sed neque vehicula rhoncus. Nam faucibus pharetra nisi.
      Integer at metus. Suspendisse potenti. Vestibulum ante
      ipsum primis in faucibus orci luctus et ultrices posuere
      cubilia Curae; Proin quis eros at metus pretium elementum.
    </div>
    <div id="display"></div>

    <script src="../js/jquery-1.11.3.min.js"></script>
    <script>
      function displayDimensions() {
        $('#display').html(
          $('#test-subject').width() + 'x' +
          $('#test-subject').height()
        );
      }

      $(window).resize(displayDimensions);
      displayDimensions();
    </script>
  </body>
</html>
```

使用虚拟文本声明测试主题

显示区域的尺寸

定义显示测试主题宽度和高度的函数

建立 resize 处理器调用显示函数

调用函数来显示初始值

110

除了提供方便的width()和height()方法外，jQuery还提供了获取尺寸的类似方法，如表5.1所示。

表 5.1 其他与 jQuery 尺寸相关的方法

方 法	描 述
<code>innerHeight()</code>	返回第一个匹配元素的内部高度，它不包含边框，但是包含内部边距。 返回值的类型为 Number，但如果 jQuery 对象为空，则此时返回 null。 如果返回 Number，则表示内部高度数据的像素单位
<code>innerHeight(value)</code>	通过 value 来设置元素的内部高度。value 可以是 String、Number 或者 Function。默认的单位是 px。如果设置的是函数，就会为 jQuery 对象中的每个元素调用该函数。该函数接受两个值：元素的索引位置及当前的内部高度值 在函数内，this 代表当前的元素。函数的返回值作为当前元素的新的内部高度值
<code>innerWidth()</code>	与 <code>innerHeight()</code> 类似，不同的是它返回的是匹配元素的宽度数据（排除边框但包含内边距）
<code>innerWidth(value)</code>	与 <code>innerHeight(value)</code> 类似，不同的是它设置的是内部匹配元素的宽度
<code>outerHeight</code> <code>([includeMargin])</code>	与 <code>innerHeight()</code> 类似，不同的是它返回的是匹配元素的外部高度，包含边框和内部边距。includeMargin 参数控制是否包含外边距
<code>outerHeight(value)</code>	与 <code>innerHeight(value)</code> 类似，不同的是它的值用来设置匹配元素的外部高度
<code>outerWidth</code> <code>([includeMargin])</code>	与 <code>innerHeight()</code> 类似，不同的是它返回的是匹配元素的外部宽度，包括边框和内部边距。includeMargin 参数控制是否包含外边距。
<code>outerWidth(value)</code>	与 <code>innerHeight(value)</code> 类似，不同的是其值用来设置匹配元素的外部宽度

111

还没有结束呢！jQuery 还提供了与定位和滚动元素相关的简化操作方法。

定位和滚动

jQuery 提供了两个方法来获取元素的位置。两个方法都返回包含两个属性 top 和 left 的 JavaScript 对象，表示元素的 top 和 left 值。

两个方法使用不同的起点来计算相对值，其中 offset() 方法返回文档的相对位置。

offset()方法语法(1)

offset()

返回集合当中第一个元素相对于文档的当前坐标。

参数

无

返回

返回与文档相关的描述位置对象，包含 left 和 top 属性，以像素为单位。

这个方法可以用来设置当前元素的坐标。

offset()方法语法(2)

offset(coordinates)

设置集合中所有元素的当前坐标(以像素为单位), 相对于文档。

参数

coordinates(Object|Function) 包含top和left属性的对象, 表示集合中元素的坐标数据。如果设置的是函数, 则会为集合中的每个元素调用函数, 当前元素作为函数上下文(this)。可以传递两个值: 元素的索引及包含top和left的对象。

函数的返回值会作为设置的新值。

返回

112 jQuery集合。

另外一个方法就是position(), 它可以返回元素最近的偏移父辈(offset parent)的坐标。偏移父辈指的是元素最近的祖先元素, 并且拥有明确的位置规则, 比如relative、absolute或者fixed的定义。position()的语法如下。

position()方法语法

position()

返回匹配集合元素中第一个元素的偏移父辈位置(像素)。

参数

无

返回

返回与偏移父辈相关的描述位置的對象, 包含left和top属性, 以像素为单位。

offset()和position() 只能用在可见元素上。

除了元素的位置, jQuery还提供了设置元素滚动条位置的方法。表5.2描述了这些方法是如何处理可见和隐藏元素的。

表 5.2 jQuery 滚动条控制方法

方 法	描 述
scrollLeft()	返回匹配元素的滚动条的水平位置 返回值的类型是 Number, 但如果 jQuery 对象为空, 则返回 null。如果返回数字, 则表示位置的值, 以像素为单位
scrollLeft(value)	设置所有匹配元素的水平滚动条的位置为 value 表示的像素单位。此方法返回它调用的 jQuery 集合
scrollTop()	与 scrollLeft()方法类似, 不同的是它返回的是垂直位置
scrollTop(value)	与 scrollLeft(value)类似, 不同的是它设置的是匹配元素的垂直位置

学习了如何使用jQuery来设置水平和垂直的滚动条位置后，就来看一个例子。

假设页面中有一个元素的id为elem，然后把位置移到页面的顶部靠左的位置。这个点的坐标是(0, 0)，意味着必须把left和top设置为0。为了实现这个效果，现在需要编写几行代码：

```
setTimeout(function() {  
    $('#elem').offset({  
        left: 0,  
        top: 0  
    });  
}, 1000);
```

113

现在来讨论几种不同的修改元素内容的方法。

5.2 设置元素内容

Setting element content

当要修改元素内容的时候，可以使用许多不同的方法。具体采用哪种方法，就取决于想要修改什么类型的内容。如果想修改的内容不需要解析成HTML标签，那么可以根据浏览器的情况选择使用textContent或者innerText的属性。

再说一次：jQuery提供了大量方法让我们可以避免处理底层浏览器兼容性的问题。

5.2.1 替换 HTML 或文本内容

第一个方法就是html()，它允许查询元素的HTML内容，正如我们看到的，和其他的jQuery方法一样，当使用参数的时候，表示要设置元素的内容。

以下是关于获取元素HTML内容的介绍。

html()方法语法(1)

html()

获取匹配集合中第一个元素的HTML内容。

参数

无

返回

返回匹配的元素的HTML内容。

下面是关于如何设置匹配元素的HTML内容的介绍。

html (content)

为匹配的元素设置传入的HTML代码块内容。

参数

content(String|Function) HTML代码块会设置为元素的内容。如果传递的是函数，就会为集合中的每个元素调用函数，当前元素作为函数执行的上下文(this)。函数接受两个值：元素索引和存在的内容。函数返回值作为新的内容。

返回

jQuery集合。

假设在页面里有如下元素定义：

```
<div id="message"></div>
```

我们就要调用一个编写的函数，当它结束的时候向用户展示一些内容。可以使用下面语句来实现：

```
$('#message').html('<p>Your current balance is <b>1000$</b></p>');
```

这行代码会导致前面的元素更新为下面的内容：

```
<div id="message"><p>Your current balance is <b>1000$</b></p></div>
```

这个例子里，传递给方法的标签会作为HTML处理。总的余额(balance)会作为粗体字显示。

除了设置HTML内容，也可以获取或者设置元素的文本内容。在无参使用text()方法的时候，返回的就是元素的文本。例如，假设有如下HTML代码：

```
<ul id="the-list">
  <li>One</li><li>Two</li><li>Three</li><li>Four</li>
</ul>
```

这条语句

```
var text = $('#the-list').text();
```

结果就是变量text赋值为OneTwoThreeFour。注意，如果元素之间有空格或者新线(比如和之间)，则会包含到结果字符串中。

此方法的语法如下。

text()方法语法(1)**text ()**

获取匹配元素集合的每个元素的内容，包含其后代节点。

text()方法语法(1)

参数

无

返回

包含所有文本内容的字符串。

也可以使用`text()`方法来设置jQuery对象中元素的文本内容。具体语法格式介绍如下。

◀ 115

text()方法语法(2)

`text(content)`

设置集合中所有元素的内容为传递的参数值。如果参数包含尖括号(<和>)或者(&), 那么这些符号将会被等价的HTML实体取代。

参数

`content(String|Number|BooleanFunction)` 要设置集合中每个元素内容的文本。当内容类型为Number或Boolean时, 它就会转换为字符串表示形式。任意的尖括号都会替换为HTML实体。如果传递的是函数, 则会为集合中的每个元素调用函数, 并且当前元素作为函数上下文(this)。函数接受两个值: 元素索引及当前的文本。函数返回值用来作为新内容。

返回

jQuery集合。

使用这些方法来修改元素的HTML或者文本内容, 都会导致之前的内容被覆盖, 所以请小心使用这些方法。jQuery并不限制这些方法, 所以来看看其他的操作。

5.2.2 移动元素

操作页面元素无须重新加载页面的方式为我们提供开发生态和交互式网页的可能性。我们已经介绍jQuery如何创建DOM元素。这些可以使用不同的方法附加到元素上, 而且可以移动(复制及移动)现有的元素。

为了在现有内容后追加内容, 可以使用`append()`方法。

append()方法语法

`append(content[, content, ..., content])`

把参数内容追加到所有匹配的元素内容后。这个方法接受任意个参数, 最少一个。

参数

`content(String|Element|jQuery|Array|Function)` 字符串、DOM元素、DOM元素的数组或者jQuery对象。如果传递的是函数, 则会为集合中的每个元素调用函数, 并且当前元素作为函数上下文(this)。函数接受两个值: 元素索引及当前的文本。函数返回值用来作为追加的内容。

返回 (content)

jQuery集合。

我们来看一个使用此方法的例子。看下面的简单例子：

```
116 > $('p').append('<b>some text<b>');
```

这条代码会把HTML代码段追加到页面p元素的现有内容后面。

一个更复杂的例子是，区分DOM元素并追加内容。看下面的代码。

```
$('p.append-to').append($('a.append'));
```

这行代码会把所有具备样式append的a元素移到所有具备样式append-to的段落元素内容的末尾。如果有多个目标要操作(append()方法要处理多个对象)，那么初始元素会被复制多次，然后追加到现有内容上。所有情况下，初始元素会从最初的位置移除。

如果目标确定，这个操作语义上就是一个移动move；最初的来源元素会被删除，然后追加到目标列表子元素的尾部。

思考下面的HTML代码：

```
<a href="http://www.manning.com" class="append">Text</a>
<p class="append-to"></p>
```

通过运行前面的代码，可以获取下面的HTML标签结果：

```
<p class="append-to">
<a href="http://www.manning.com" class="append">Text</a>
</p>
```

对于多个目标，这个操作也可以作为复制并且移动操作，创建所有初始元素的副本，这样可以给每个目标的子元素追加新元素。

对于集合，可以引用特定的DOM元素，代码如下：

```
$('#p.appendToMe').append(someElement);
```

另外一个使用此方法的例子如下：

```
$('#message').append(
  '<p>This</p>',
  [
    '<p>is</p>',
    $('#<p>').text('my')
  ],
  $('#<p>text</p>')
);
```


在这个代码里可以看到`append()`方法如何管理多个参数，而且每个参数可以是不同的类型（字符串、数组或者jQuery对象）。运行代码的结果就是会有一个ID为`message`的元素，包含四个段落，它们组成的句子就是“This is my text.”

虽然添加内容到元素内容的末尾是常见的需求，但有时候会要求添加一个列表项到另外一个列表末尾，或添加一行内容到一个表末尾，或者添加新元素到文档的末尾。还有时候也许需要添加新的或者现有的元素到最前面。

当出现这种需求时，用`prepend()`方法就可以处理这种问题。

`prepend()`方法语法

`prepend(content[, content, ..., content])`

对于所有匹配的元素，插入参数到内容的最前面。这个方法接受任意数量的参数，最少为一个。

参数

`content` 与`append()`方法的参数`content`类似，不同之处是插入每个元素内容到头部。

返回

jQuery集合。

有时插入元素的位置既不是元素内容的开头也不是元素的结尾。jQuery允许放置内容到DOM中的任意位置，并区分插入目标元素的位置。

不出意料，`before()`和`after()`两个方法就是用来处理这个问题的。现在对它们的语法描述应该很熟悉了。

`before()`方法语法

`before(content[, content, ..., content])`

把传递的参数插入目标元素的前面。目标元素必须是DOM元素的一部分。这个方法接受任意数量的参数，至少有一个。

参数

`content` 与`append()`方法的参数`content`一样，不同之处是插入参数到每个匹配元素的前面。

返回

jQuery集合。

`after()`方法语法

`after(content[, content, ..., content])`

把传递的参数插入目标元素的后面。目标元素必须是DOM元素的一部分。这个方法接受任意数量的参数，至少有一个。

参数

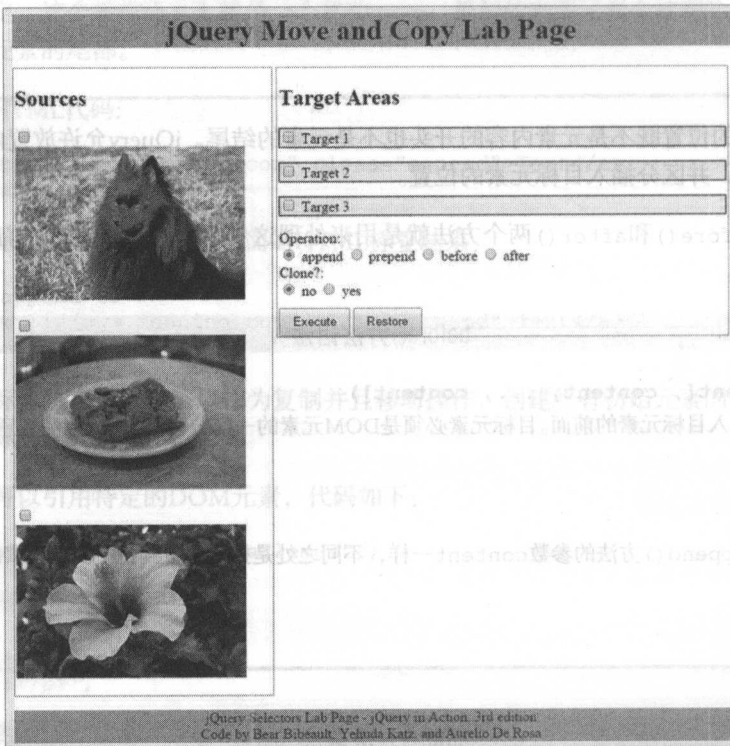
content 与 append() 方法的参数 content 类似，不同之处是插入参数到每个匹配元素的后面。

返回**118 jQuery 集合。**

这些方法对于高效地操作页面元素非常重要。这里提供了移动和复制的动手实验页面，这样就可以练习操作，直到掌握这些方法。这个实验页面可以在 `chapter-5/lab.move_and.copy.html` 中找到，初始显示如图 5.3 所示。

左边包含三个图片，作为移动和复制的目标。点击勾选框选择一个或者多个图片。

移动/复制的目标区域位于页面的右侧，也可以通过复选框选择。底部的控件允许选中四项操作：`append`（追加末尾）、`prepend`（前插头部）、`before`（前面）、`after`（后面）（现在先忽略“clone”，后面再介绍）。



119 图 5.3 移动和复制实验页面允许我们监测 DOM 操作方法

点击“Execute（执行）”按钮就会把选中的图片移动到目标区域里。当想恢复原样时，可以使用“Restore（恢复）”按钮重新运行实验页面。



现在运行追加内容实验。选择狗的图片 and 选择Target 2(目标2), 再选择追加append(操作)。点击“Execute”按钮, 结果如图5.4所示。

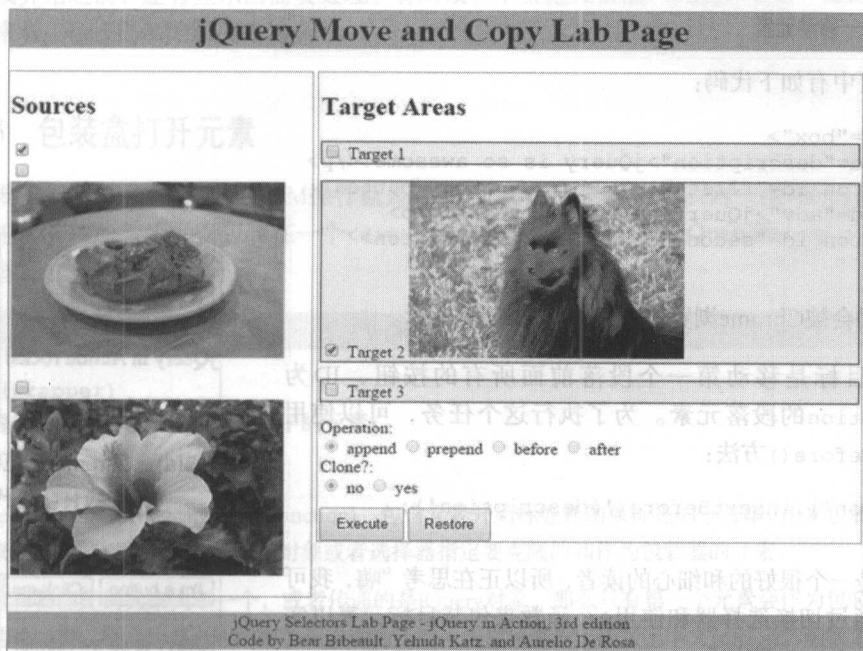


图 5.4 小狗图片追加到 Target 2 的尾部

使用移动和复制的实验页面来尝试各种不同的元素、目标和四项操作的组合, 直到完全理解这些操作。

有时候, 它也许可以让代码可读性更强, 如果能反转传递给操作的元素顺序。如果想要移动或者复制一个元素从一个地方到另外一个地方, 可行的办法就是包装源元素(而不是目标元素), 而且在方法参数里指定目标元素。jQuery 提供除 `appendTo()`、`prependTo()`、`insertBefore()` 和 `insertAfter()` 这四项操作以外, 还提供更多的方法来反转源元素和目标元素的顺序。描述如表 5.3 所示。

120

表 5.3 移动 DOM 元素的其他方法

方 法	描 述
<code>appendTo(target)</code>	把集合中所有匹配的元素插入目标元素的内容尾部。目标(target)参数可以是包含选择器的字符串、HTML 字符串、DOM 元素、DOM 元素的数组或者 jQuery 对象。方法返回调用的 jQuery 对象
<code>prependTo(target)</code>	与 <code>appendTo(target)</code> 方法类似, 不同点在于是插入到内容的开头
<code>insertBefore(target)</code>	与 <code>appendTo(target)</code> 类似, 不同点在于把元素插入指定目标的前面
<code>insertAfter(target)</code>	与 <code>appendTo(target)</code> 类似, 不同点在于把元素插入指定目标的后面

为了学习这些方法，我们下面来看几个例子。

例子 #1——移动元素

假设网页中有如下代码：

```
<div id="box">
  <p id="description">jQuery is so awesome!</p>
  <button id="first-btn">I'm a button</button>
  <p id="adv">jQuery in Action rocks!</p>
  <button id="second-btn">Click me</button>
</div>
```

这段代码会被Chrome浏览器渲染，如图5.5(a)所示。

第一个目标是移动第一个段落前面所有的按钮，ID为description的段落元素。为了执行这个任务，可以使用insertBefore()方法：

```
$('button').insertBefore('#description');
```

因为你是一个很好的和细心的读者，所以正在思考“嗨，我可以使用通过切换选择器和使用\$()函数来包装目标元素以实现调用before()方法”。

恭喜，你做对了！前面的语句可以等价转换为下面的代码：

```
$('#description').before($('button'));
```

一旦执行，无论运行的是前面两者中的哪条代码，页面都会更新为图5.5(b)所示的样子。

可以在JS Bin (<http://jsbin.com/ARedIWU/edit?html,js,output>)或者chapter-5/moving.buttons.htm中找到例子页面(只要页面被加载，按钮就会被删除)。现在来看一些更复杂的例子。

例子 #2——拷贝并复制内容

假设有与之前例子一样的HTML标签代码，想创建一个新的段落元素，它的内容等于div内两个段落内容的合集，然后把它放到这个div的右边。新创建的段落内容将会是“jQuery is so awesome! jQuery in Action rocks!”可以通过本章前面介绍的两个方法text()和after()来实现。满足这个需求的代码实现如下：

```
var $newParagraph = $('<p></p>').text(
  $('#description').text() + ' ' + $('#adv').text()
);
$('#box').after($newParagraph);
```

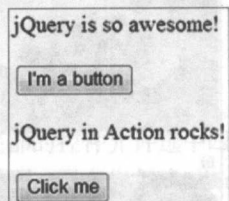


图 5.5(a) Chrome 浏览器渲染的 HTML 代码

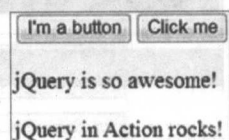


图 5.5(b) 在执行完代码后，Chrome 浏览器渲染的 HTML 代码

这两个例子应该可以证明，发现的方法越多，获得的能量就越大。而且，现在还没有结束呢！在继续介绍之前，还有些东西需要处理。有时候，不会把元素插入其他元素里，而恰恰相反。下面来看jQuery提供的解决办法。

5.2.3 包装盒打开元素

经常要执行另外一种类型的DOM操作就是把元素(或者元素集合)包装到一些HTML标签中。若想把所有的超链接元素包装在一个<div>元素里，则可以通过使用jQuery的wrap()方法来实

wrap()方法语法

wrap(wrapper)

使用提供的参数来包装jQuery对象中的元素。

参数

wrapper(String|Element|jQuery|Function) 包含元素开始标签和结束标签的字符串，用来包装匹配的集合元素。参数可以是元素、jQuery对象或者选择器指定要克隆的和作为包装器的元素。

如果选择器匹配的元素大于一个，或者传递的是jQuery对象，那么只有第一个元素会作为包装器。如果提供的是函数，则会为集合中的每个元素调用函数，传递元素作为函数上下文(this)。函数返回值可以是HTML代码块或者jQuery对象，用来包装当前元素。

返回

jQuery集合。

为了掌握这个方法的使用过程，假设有如下代码：

```
<a class="surprise">A text</a>
<a>Hi there!</a>
<a class="surprise">Another text</a>
```

为了使用具备样式hello的<div>来包装具各样式为surprise的超链接元素，可以编写如下代码：

```
$('a.surprise').wrap('<div class="hello"></div>');
```

运行这行代码生成的结果如下：

```
<div class="hello"><a class="surprise">A text</a></div>
<a>Hi there!</a>
<div class="hello"><a class="surprise">Another text</a></div>
```

如果想要用页面中假想的第一个div的克隆元素来包装超链接元素，则可以这样编写代码：

```
$('a.surprise').wrap($('div:first'));
```


或者这样编写代码：

```
$('#a.surprise').wrap('div:first');
```

记住，这个例子中div的内容也会被克隆，而且用来包装a元素。

当jQuery对象中有多个元素要收集时，wrap()方法会单独处理每一个元素。如果想把jQuery对象里的所有元素作为一个单位进行包装，那么可以使用wrapAll()方法来代替。

wrapAll()方法语法

wrapAll(wrapper)

使用提供的参数作为一个单元来包装匹配集中的元素。

参数

wrapper 与wrap()方法的参数wrapper一样。

返回

jQuery集合。

jQuery 3:bug 修复

当传递函数给wrapAll()方法一个bug时，jQuery 3就会修复它。在jQuery 3之前的版本，当传递函数给wrapAll()时，它会把匹配集中的每个元素单独进行包装而不是作为一个单元进行包装。它的行为与传递函数给wrap()方法一样。

除了修复这个bug外，在jQuery 3中，函数只会调用一次，它不会传递jQuery对象中元素的索引。最后，函数上下文(this)现在引用的是匹配集中的第一个元素。

123

有时不想在一个匹配集合里包装元素，而是想包装它们的内容。对于这种情况，可以使用wrapInner()方法。

wrapInner()方法语法

wrapInner(wrapper)

使用参数来包装匹配集中元素的内容，包括文本节点。

参数

wrapper 与wrap()方法的参数wrapper一样。

返回

jQuery集合。

与wrap()方法相反的操作就是删除子元素的父元素，即使用unwrap()方法。

unwrap()

删除集合内元素的父元素。子元素与其他兄弟节点可以替换DOM中的父元素。

参数

无

返回

jQuery集合。

jQuery 3: 添加参数

jQuery 3添加了一个可选selector给unwrap()方法。可以传递一个包含选择器表达式的字符串来匹配父元素。假如匹配到了父元素，子元素就会解开包装；否则操作无效。

在继续讨论之前，我们来看这个方法使用的例子。

如何包装表单的 LABEL-INPUT 和 LABEL-TEXTAREA 元素

假设有如下联系表单代码：

```
<form id="contact" method="post">
  <label for="name">Name:</label>
  <input name="name" id="name" />
  <label for="email">Email:</label>
  <input name="email" id="email" />
  <label for="subject">Subject:</label>
  <input name="subject" id="subject" />
  <label for="message">Message:</label>
  <textarea name="message" id="message"></textarea>
  <input type="submit" value="Submit" />
</form>
```

124

想要在div里包装每个label-input或者label-textarea元素，就要设置div的样式为field。可以定义样式field如下：

```
.field
{
  border: 1px solid black;
  margin: 5px 0;
}
```

包装标签对我们并不意味着我们并不想单独包装每个元素。为了实现目标，就需要使用第1章学习的知识。这也可以测试我们是否深入理解了之前介绍的概念或者需要快速温习一下。一种可行的解决方案如下：

选择所有表单的<input>和<textarea>元素后单独处理

```

$('input, textarea', '#contact').each(function(index, element) {
    var $this = $(this);
    $this
        .add($this.prev('label'))
        .wrapAll('<div class="field"></div>');
});

```

缓存只包含当前元素集合

只有当它是<label>时才会添加到前置同级元素集合里

包装<div>内的对

为了执行练习代码，需要选择<form>标签里的所有<input>和<textarea>元素（为了简化，我们忽略了<select>标签），而且要找到一种方式与<label>关联。然后，还要单独处理每个标签对。因此，在选择所有需要的元素后，还要使用第3章介绍的each()方法。在传递给它的匿名函数里，使用\$()包装当前元素，并存储在一个变量里，因为我们将两次使用它。然后添加这个元素到集合中。它的前置兄弟元素必须是<label>标签元素，此时有了两个需要的元素集合，再包装到<div>元素中。耶！任务完成了！

图5.6(a)展示了运行代码之前的<form>，图5.6(b)展示了运行之后的结果。

图 5.6(a) 执行代码之前的表单

图 5.6(b) 执行代码后的表单

注意每个label-input或者label-textarea对是如何被黑色边框包围的，证明它们已经被<div>包围了。

假如想进一步操作这个例子，可以在JS Bin (<http://jsbin.com/lrUhfAg/edit?html,css,js,output>) 和文件chapter-5/wrapping.form.elements.html中找到。

目前为止，我们已经学习了如何执行各种操作。现在是时候学习如何从DOM中删除元素了。

5.2.4 删除元素

有时候，想删除那些不再需要的元素。如果想删除一个集合的元素和它们所有的内容，则可以使用remove()方法，其语法说明如下。

remove()方法语法

remove([selector])

从页面中删除集合中的所有元素及其内容，包括事件侦听器及任意存储的数据。

参数

selector(String) 可选选择器参数，筛选集合中被删除的元素。

返回

jQuery集合。

注意，与许多其他的jQuery方法类似，这个方法的结果也是集合。DOM中删除的元素仍然被这个集合引用（而且当前状态垃圾回收器无法回收），可以继续使用其他的jQuery方法来操作，包括appendTo()、prependTo()、insertBefore()、insertAfter()等。但是，被删除元素存储的数据和事件侦听器已经丢失。

如果想从DOM中删除元素，且希望保留绑定的事件和数据（使用data()方法添加的数据），那么可以使用detach()方法。

detach()方法语法

detach([selector])

从页面DOM里删除所有元素及其内容，但是保留绑定的事件和jQuery数据。

参数

selector(Selector) 可选参数选择器，字符串进一步过滤要删除的元素。

返回

jQuery集合。

detach()方法是删除可能将来要放回DOM元素里的事件和数据的推荐方法。典型的场景就是，从DOM中拉出一些元素，应用一些修改，然后再把它们放回到DOM里。这样做会改进代码的性能，因为修改一个分离的元素比应用所有修改到一个元素或者DOM的多个元素上更快。

为了完整清空DOM元素的内容且保留元素，可以使用empty()方法。其语法如下。

empty()方法语法

empty()

删除集合里所有DOM元素的内容。

参数

无

返回

jQuery集合。

126

这个方法在处理使用Ajax获取的外部内容时非常有用。假设获取了一些新的内容，而且现在需要把它添加到页面的一个ID为content 的<div>里，可以使用下面的代码来实现：

```
var newContent = '<p>Wow, this new content is awesome!</p>';  
$('#content')  
  .empty()  
  .html(newContent);
```

从外部资源获得的内容
删除内容
把获取的内容作为 HTML 注入此元素中
查询元素

记住，小心使用html()方法注入页面中的外部内容，因为添加的内容也许是XSS(跨站脚本攻击(cross-sitescripting))和CSRF(跨站请求伪造(cross-site request forgery))。

127 删除元素很简单，但是有时需要克隆一些元素。

5.2.5 克隆元素

一种可行的方法是拷贝DOM元素，然后附加到DOM树的其他地方。jQuery提供了更简便的clone()方法。

clone()方法语法

clone([copyHandlersAndData[, copyChildrenHandlersAndData]])

jQuery集合中的元素的深度拷贝，返回一个新的元素集合。元素和子元素全部被复制。事件及数据可以根据copyHandlersAndData参数的设置进行拷贝。

参数

copyHandlersAndData(Boolean) 如果为true，则事件处理器和数据都会被复制；如果为false，则不会被复制。

copyChildrenHandlersAndData(Boolean) 如果为true，则复制所有子元素的事件处理器和数据；如果忽略，则只提供第一个参数，使用相同的值；如果为false，则不会被复制。

返回

新创建的jQuery集合。

对于已有的元素，使用clone()方法没有什么用处，除非对复制的元素再做些处理。通常来说，一旦生成包含克隆的集合，应用另外一个jQuery方法就会把它们贴在DOM中。例如，

```
$('#img').clone().appendTo('fieldset.photo');
```

复制所有的图片(img)元素，然后把它们附加到使用了样式photo的元素fieldset中。

一个更有意思的例子如下：

```
$('#ul').clone(true).insertBefore('#here');
```

这个方法链执行一个类似的操作，但是克隆操作的目标是所有的ul元素被复制，包括事件处

理器和数据。此外，因为`clone()`方法也会克隆子元素，而且任意一个`ul`元素都会包含许多的`li`子元素，所以要确保没有丢失任何信息。

若忽略第二个参数，但是指定了第一个参数，所有子元素的数据和事件处理器也都会被复制。

在学习下一个主题之前，我们来讨论最后一个例子。假设有一个超链接的集合，包含图片。想要复制所有的元素，然后放到页面第一个`div`元素里。此外，还想保留时间处理器和数据，但是不想要内部的图片。这个任务可以使用下面的代码来实现。

```
$('#a').clone(true, false).appendTo('div:first');
```

这行代码展示了如何使用方法的可选参数。



为了实际查看克隆操作，返回移动和复制实验页面(Move and Copy Lab Page)。在执行Execute按钮上面是一对单选按钮(`radio button`)，它们允许指定克隆操作作为页面操作的一部分。当选择Yes克隆单选按钮，资源就会被克隆，`append()`、`prepend()`、`before()`及`after()`方法就会被执行。

打开克隆按钮，重复之前的实验。注意：初始资源是不受影响的。

也可以插入、删除、复制元素。虽然可将这些操作联合起来，但是执行更高层次的操作比如替换会更简单。为什么呢？因为不需要这么复杂的操作。

5.2.6 替换元素

当想要使用新的元素来替换已有的元素，或者移动现在的元素来替换另外一个元素时，jQuery提供了`replaceWith()`方法。

replaceWith()方法语法

replaceWith(content)

使用指定的内容来提供匹配的元素。

参数

`content(String|Element|Array|jQuery|Function)` 要替换的内容包含HTML代码块，或者DOM元素，或者DOM元素的数组，或者jQuery对象。如果设置为函数，则函数会调用集合中的每个元素，元素会作为函数上下文(`this`)，无需参数。函数返回值作为新的内容。

返回

包含替换的jQuery集合。

为了明白这个方法的用处，我们来看一个例子。假设有如下的代码：

```



```

我们想要替换所有包含alt属性的图片元素，一次一个，使用span元素。后者会导致图片的alt属性被替换。使用each() 和replaceWith() 方法，可以这样做：

```
129 > $('img[alt]').each(function() {  
    $(this).replaceWith('<span>' + $(this).attr('alt') + '</span>');  
});
```

使用each() 方法可以在集合中的每个元素上进行迭代，而replaceWith() 方法是使用span元素来替换图片元素，结果标签如下：

```
557 <span>A ball</span>  
557 <span>A blue bird</span>  
557 
```

这个例子又一次展示了使用jQuery操作DOM元素是多么简单。



如果想要对抛弃的元素再做处理，可用replaceWith() 方法返回包含删除元素的jQuery集合。作为练习，思考假如要把DOM中删除的元素重新附加到元素的其他地方，怎么办？

当传递现有元素作为replaceWith() 方法的参数时，它就会从DOM中的原始位置删除，重新添加替换目标元素。如果包含多个目标，则最初的元素会被克隆多次。

有时，使用replaceWith() 方法反转元素的顺序非常方便，所以可以使用特殊的选择器来指定要替换的元素。我们已经学习过这种互补性的方法，比如append() 和appendTo()，允许指定元素的顺序，这是非常有意义的。

与之相似的是，replaceAll() 方法的镜像方法replaceWith() 允许执行类似的操作。但是，在这个例子里，要替换的元素是通过选择器参数来指定的，因此并不是方法调用的最初元素。

replaceAll()方法语法

replaceAll(target)

使用传递的target替换匹配的每一个元素。

参数

target(String|Element|Array|jQuery) 选择器表达式、DOM元素或者DOM元素的数组，或者指定了要替换元素的jQuery集合。

返回

包含插入元素的jQuery集合。

与replaceWith() 很像，replaceAll() 返回一个jQuery集合，但是它不包含被替换的元素，而是包含替换元素。被替换的元素已经丢失，无法再进行其他操作。记住要选择的替换方法。

130 > 根据replaceAll() 方法的描述，可以使用下面的代码实现前面例子的目标：


```
$( 'img[alt]' ).each( function() {
    $( '<span>' + $( this ).attr( 'alt' ) + '</span>' ).replaceAll( this );
});
```

注意，如何反转传递给 `$()` 和 `replaceAll()` 的参数。

现在已经讨论了通用的处理DOM元素的方法，下面简要看看如何处理特定的元素类型：表单元素。

5.3 处理表单元素的值

Dealing with form element values

因为表单元素拥有特殊的属性，所以jQuery内核包含许多便捷的方法来处理这些问题，比如：

- 获取和设置元素的值。
- 序列化。
- 基于表单属性来选择元素。

什么是表单元素

当使用表单元素词汇时，是指出现在表单中的元素，拥有name和value特性，而且当表单提交的时候它的值会作为HTTP请求参数发送给服务器。

让我们来看一个最常见的操作表单元素的例子：获取元素的值。jQuery的 `val()` 方法处理这种需求，返回jQuery对象里第一个表单元素的value值，它的语法如下。

val()方法语法(1)

val()
返回jQuery集合中的第一个元素的值。如果第一个元素是<select>，并且没有选中项目，方法就会返回null。如果元素是多选框(<select>包含多个指定的属性)，而且至少有一个被选中，返回值就是包含选项的数组。

参数

无

返回

获取的值。

这个方法虽然相当有用，但是它也包含许多限制，需要多加小心。如果集合中的第一个元素不是表单元素，就会返回空字符串，对此有些人可能会产生误解。这个方法不会区分勾选框和单选按钮的checked或者unchecked状态，会直接返回元素的value值，无论它们是否被选中。

对于单选按钮，jQuery选择器可以与val()方法结合使用。思考一个表单包含一组单选按钮的名字都是radio-group（单选按钮名字一样）的情况。表达式如下：

```
$('#input[type="radio"][name="radio-group"]:checked').val();
```

这个表达式会返回单个选中的单选按钮（radio button）的值（若没有选中，就返回undefined）。这比循环遍历所有的按钮来查找选中的元素简单得多，不是吗？

因为val()只考虑集合中的第一个元素，所以对于检查一组勾选框没有作用，因为可能选中多个控件。jQuery会提供相应的解决办法。考虑下面的代码：

```
var checkboxValues =  
    $('#input[type="checkbox"][name="checkboxgroup"]:checked').  
        map(function() {  
            return $(this).val();  
        })  
    .toArray();
```

虽然val()方法对于获取单个表单控件的值非常有用，但是，如果要获取整个集合元素的值，就必须通过表单提交方式来完成，最好使用serialize()或者serializeArray()方法（将在第10章介绍）。

另外一个常见的操作就是设置表单元素的值。val()方法也可以用来为元素赋值，它的语法说明如下。

val()方法语法(2)

val (value)

为匹配元素设置参数value指定的值。如果提供的是数组值，则会导致匹配传入数组values集合中的任意复选框、单选框或者select元素项目被选中。

参数

value(String|Number|Array|Function) 指定集合中每个元素value属性要赋值的值。数组元素的值会用来决定哪个元素被选中。如果参数传递的是函数，就会为集合中的每个元素调用函数，当前元素作为函数调用上下文(this)。函数接收两个值：元素的索引和当前值。函数返回值作为新的赋值。

返回

jQuery集合。

正如方法说明描述的，val()方法会导致勾选框或单选按钮，或者<select>的项目被选中。如下面的代码：

```
132) $('#input[type="checkbox"], select').val(['one', 'two', 'three']);
```

它会查找页面上与输入值"one"、"two"或"three"匹配的所有复选框（checkbox）和下拉列表select元素。任意找到的匹配元素都会被设置为选中。对于没有multiple属性的select元素，只有第一个元素被选中。在之前的代码里，只有包含one的项目被选中，因为传递给val()

方法的字符串"one"在字符串"two"和"three"前面。这让val()的作用远远超出基于文本的表单元素的。

5.4 总结

Summary

学习完本章的知识以后，可以在网页上复制元素、移动元素、替换元素甚至删除元素，也可以追加、前插或者包装元素。此外，讨论了如何管理表单元素的值，所有技巧都可以用来实现强大而简洁的逻辑功能。

掌握了这些知识以后，可以准备学习一些更高级的主题了。现在就从页面上典型、复杂的事件处理工作开始吧。

133

6.1 理解浏览器事件模型

很久以前，人们就开始思考浏览器中事件的处理问题。网景公司(Netscape Communications Corporation)在其Netscape Navigator浏览器中引入了事件处理模型。这个模型中，事件对象(Event Object)是事件处理的核心。事件对象(Event Object)是一个对象，它包含了事件的所有信息。事件对象(Event Object)的创建是由浏览器完成的。当用户与网页交互时，浏览器会创建一个事件对象，并将其传递给事件处理函数。事件处理函数可以访问事件对象的属性，并根据需要执行相应的操作。事件对象(Event Object)的属性包括事件的类型、事件的源、事件的坐标等。事件对象(Event Object)的创建是由浏览器完成的。当用户与网页交互时，浏览器会创建一个事件对象，并将其传递给事件处理函数。事件处理函数可以访问事件对象的属性，并根据需要执行相应的操作。

事件本质

Events are where it happens!

本章内容

- 浏览器实现的事件模型。
- jQuery事件模型。
- 绑定事件处理器到DOM元素。
- 事件委托。
- 命名空间划分事件。
- Event对象实例。
- 触发脚本控制下的事件处理程序。
- 注册主动事件处理器。

与其他的GUI管理系统一样，通过HTML网页展示的接口也是异步(asynchronous)和事件驱动(event-driven)的，即使使用HTTP协议来传输给客户端浏览器，HTTP协议本质上也是同步的。无论使用Java Swing、X11或.NET，或者HTML和JavaScript来开发GUI，程序运行的步骤都是类似的。

1. 设置用户界面(UI)。
2. 等待有趣的事情发生。
3. 做出相应的反应。
4. 返回第2步。

第1步就是显示(display)用户界面，其他定义软件的行为(behavior)。在网页中，浏览器负责处理显示工作，处理HTML和CSS代码。页面里的脚本负责界面的行为，当然它也可以修改界面。

这个脚本组成了事件侦听器(event listeners)，也称为事件处理器(event handlers，技术上有点差别)。这些事件通过系统生成(比如计时器(timer)或者完整的异步请求)，但通常是由用户行为(比如用户移动鼠标、点击鼠标按钮、通过键盘输入文本，或者触摸手势)导致的。如果没有处理这些事件的能力，WWW的能力就会大打折扣，最多也就是用来显示图片而已。

虽然HTML本身已经内部定义了一些无须脚本代码的行为(比如点击锚(anchor)标签的时候重新加载页面或者点击“submit”按钮提交页面),但是要处理各种不同的其他事件,还是需要自己来编写代码,以响应页面的用户交互行为。

本章将详细检查浏览器暴露事件时的不同处理方式,以及自己如何建立事件处理器来控制事件发生时执行的代码。此外,还会介绍不同浏览器事件模型处理事件带来的兼容性问题。然后看看jQuery如何帮助我们解决这些跨浏览器的兼容性问题。

下面先从学习浏览器暴露的事件模型开始。

你应该知道的 JavaScript!

jQuery带给Web网页的一大好处就是,不需要编写太多的脚本代码就可以实现大量脚本实现的行为。jQuery用来处理底层的交互问题,可以让我们专注于网站功能的开发。

此时,开发过程就没有这么痛苦了。我们需要的只是基本的JavaScript编程能力,理解前面章节里介绍的jQuery的例子即可。在本章和以后的章节里,必须掌握一些重要的、基本的JavaScript概念,以便高效使用jQuery库。

取决于我们的背景,可能已经熟悉了这些概念,但是一些网页开发者可能还没有牢固掌握这些知识——JavaScript的灵活性导致难以一下子掌握全部内容。在继续学习之前,应该确保我们的脑袋里已经装满了这些概念。

如果已经非常熟悉JavaScript对象(Object)和函数(Function),而且掌握了诸如函数上下文和闭包的概念,那么你可能想继续阅读接下来的章节。如果对这些概念还十分陌生,那么强烈推荐你去附录部分快速熟悉这些概念,然后再回来继续学习。

135

6.1 理解浏览器事件模型

Understanding the browser event models

很久以前,人们就开始思考浏览器处理事件的标准化问题,网景公司(Netscape Communications Corporation)在其Netscape Navigator浏览器里引入了事件处理模型。这个模型有很多个名字。你可能听说过网景事件模型(Netscape Event Model)、基本事件模型(Basic Event Model),或者非常模糊的概念浏览器事件模型(Browser Event Model),但是大部分人称其为DOM Level 0 Event Model(DOM级别0事件模型)。

注意:DOM level(DOM 级别)用来表示实现W3C DOM规范的级别需求。并没有DOM Level 0,这是采用非正式的方式来描述DOM Level 1之前的要求。

在2000年11月制定DOM Level 2规范之前,W3C并没有为事件处理创建标准的模型。这个模型被所有的新浏览器支持,比如IE9以上、Firefox、Chrome、Safari及Opera。IE8之前的浏览器

有自己的方式，支持DOM Level 2事件模型的功能的子集，虽然使用的是专用接口。

在向你展示jQuery如何处理这些棘手的问题之前，需要花点时间了解各种不同的事件模型机制。

6.1.1 DOM 级别 0 事件模型

DOM级别0事件模型是大多数业余网站开发者在网页中使用的模型，因为它独立于浏览器，而且简单易用。

在这个事件模型背后，事件处理器通过把函数赋值给元素的属性来声明实现。这些属性定义了不同的事件类型，例如click事件被onclick属性赋值函数处理。mouseover事件通过把函数赋值给元素的onmouseover属性来处理。

浏览器允许我们指定事件处理器函数作为DOM元素的HTML标记属性保存起来，提供简便的创建事件处理器的方式。列表6.1显示了定义事件处理器的两种方式，它可以在本书的例子代码chapter-6/dom.0.events.html里找到。

列表 6.1 声明 DOM Level 0 事件模型

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Level 0 Events - jQuery in Action, 3rd edition</title>
    <link rel="stylesheet" href="../css/main.css"/>
    <style>
      img
      {
        display: block;
        margin: auto;
      }
    </style>
  </head>
  <body>
    
    <script>
      function formatDate(date) {
        return (date.getHours() < 10 ? '0' : '') + date.getHours() +
          ':' + (date.getMinutes() < 10 ? '0' : '') + date.getMinutes() +
          ':' + (date.getSeconds() < 10 ? '0' : '') + date.getSeconds() +
          '.' + (date.getMilliseconds() < 10 ?
            '00' : (date.getMilliseconds() < 100 ? '0' : '')) +
            date.getMilliseconds();
      }
      document.getElementById('example').onmouseover = function(event) {
        console.log('At ' + formatDate(new Date()) + ' Crackle!');
      };
    </script>
  </body>
</html>
```

设置 img 元素

定义 mouseover 处理器

在控制台显示文本

在这个例子中，使用了两种风格的事件处理器声明：一种是直接在标签属性❶中声明，另外一种是在脚本控制代码标签中❷声明。在页面的主体中，定义一个图片元素，它的ID为example。使用onclick属性❸定义click事件处理器。

在<script>标签内，定义了一个日期格式化函数formatDate()，接受Date类型的对象，再返回字符串事件作为结果。然后使用JavaScript的getElementById()方法，查找引入的图片，在函数内部❹设置onmouseover属性。当鼠标滑过元素上面时触发mouseover事件，函数作为事件处理器。注意：函数只接受一个参数，后面再讨论。在函数内部，打印格式化的事件到控制台上，又使用了之前声明的函数formatDate()❺。

注意：这个例子已经介绍了 JavaScript 的一些概念。在阅读完本章之前，先了解为什么不需要手动在 DOM 标签里嵌入事件行为！

137

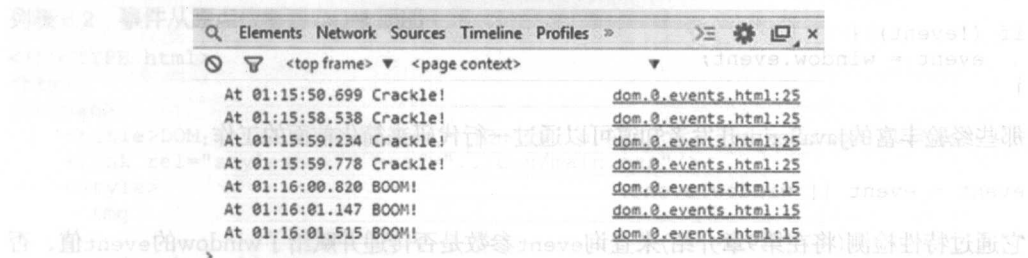


图 6.1 滑动鼠标到图片上，然后点击图片触发事件处理代码，在控制台上打印消息

在Chrome浏览器里加载网页(chapter-6/dom.0.events.html)，滑动鼠标指针到图片上几次，然后点击图片，就会看到图6.1所示的结果。

在img元素标签里声明了click事件处理器，可以使用下面的属性：

```
onclick="console.log('At ' + formatDate(new Date()) + ' BOOM!');"
```

这可能会导致你相信这段代码会成为元素的事件处理器，但事实并非如此。当通过HTML标签属性来声明处理器时，就会使用属性值来自动创建一个匿名函数体。假设imageElement引用的是一个图片元素，等价的属性声明的代码如下：

```
imageElement.onclick = function(event) {  
  console.log('At ' + formatDate(new Date()) + ' BOOM!');  
};
```

注意，属性值是如何用来生成函数体的，而且函数被创建，event参数就可以在生成的函数体内进行访问。最后，在函数内，可以使用this关键字来访问元素。虽然还没有在简单的例子里使用它，但这是我们要强调的重点知识。

再强调一次，想提醒你使用属性机制来声明DOM级别0事件处理器违反了低调的JavaScript原则。我们会很快看到JavaScript和jQuery，也可以处理浏览器的不兼容性问题，它们提供了比之前方式更好的方法来声明事件处理器。但是首先，我们要先深入了解event参数的本质。

事件实例

当激发事件处理器时，在所有兼容标准的浏览器里，一个名为Event的对象会作为第一个参数传递给处理器。再强调一次，只有IE9以上、Firefox、Chrome、Safari和Opera支持。IE8之前的浏览器可通过自己的方式把Event对象传递到全局的event属性(换句话说，window的属性)上。使用旧的脚本来保持向后的兼容性得不偿失，新版本的IE仍然保留对于window里事件的引用。不止这些，微软还保留了它独有的特性，与标准的对象合并。值得注意的是，虽然Chrome浏览器在DOM级别2事件模型规范公布很久才发布，但是它也添加了window对象中事件的引用，而且也支持IE的属性。

为了处理这些差异，在一些非jQuery的事件处理器代码里会经常看到这些语法：

```
if (!event) {  
    event = window.event;  
}
```

那些经验丰富的JavaScript开发者知道可以通过一行代码来简化前面的工作：

```
event = event || window.event;
```

它通过特性检测(将在第9章介绍)来查询event参数是否传递并赋给了window的event值，否则就赋值给它。在这行代码后，就可以访问event参数，无论它是怎么提供给处理器的。

Event实例的属性提供了大量关于事件的信息，而且正在被处理。这包括触发事件所在的元素、鼠标事件的坐标，以及键盘事件点击的键。

如果在处理旧版本的IE浏览器，不仅使用专门的方法来获取处理器的Event实例，而且使用特有的Event对象定义来代替W3C定义的标准——还必须处理对象检测的问题。因为这些异常，你应该明白开发者对于新版本的IE支持W3C标准有多开心。微软并不像传言的这么邪恶，而且现在更加开放。

为了给大家各浏览器不一致的直观感受，我们来一起看一个例子。为了获取目标元素的引用——事件被触发所在的元素——必须通过兼容标准浏览器的target属性来获取，但是在旧的IE里使用srcElement属性。要处理这种浏览器标准不一致的问题，必须使用下面的代码来做浏览器功能检测：

```
var target = event.target || event.srcElement;
```

这行代码测试是否定义了`event.target`，如果定义了，就赋值给局部变量`target`，否则就复制`event.srcElement`。需要对其其他的`event`属性采用相似的步骤。

直此为止，我们已经把触发事件的元素当成唯一事件处理器来处理，比如列表6.1中的图片元素。但是，事件是通过DOM树进行传播的。下面看看相关的知识。

事件冒泡

当DOM树的某个元素触发了某个事件时，浏览器的事件处理机制就会检查元素是否为这个特定的事件建立了处理器，如果有，就会调用。但是这并没有完全结束。

在目标元素有机会处理该事件后，浏览器的事件处理机制会检查元素的父元素是否包含此事件类型的处理器，如果有，也会调用。此刻，它的父元素被检查，再依次检查父父元素，等等，直到DOM树的顶部。因为事件处理向上传播机制就像香槟杯里的气泡一样(假设DOM树的根在顶部)，所以这个过程被称为事件冒泡(event bubbling)。

下面修改列表6.1的例子，这样就可以立即查看这个过程。

思考列表6.2，可以在chapter-6/dom.0.propagation.html中找到这个例子。

列表 6.2 事件从原点传播到 DOM 顶部

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Level 0 Bubbling - jQuery in Action, 3rd edition</title>
  <link rel="stylesheet" href="../css/main.css"/>
  <style>
    img
    {
      display: block;
      margin: auto;
    }
  </style>
</head>
<body>
  <div id="greatgrandpa">
    <div id="grandpa">
      <div id="pops">
        
      </div>
    </div>
  </div>
</body>
<script>
  function formatDate(date) {
    return (date.getHours() < 10 ? '0' : '') + date.getHours() +
      ':' + (date.getMinutes() < 10 ? '0' : '') + date.getMinutes() +
      ':' + (date.getSeconds() < 10 ? '0' : '') + date.getSeconds() +
      '.' + (date.getMilliseconds() < 10 ?
```

```

921 > 注意：脚本是如：'00' : (date.getMilliseconds() < 100 ? '0' : '') +
    date.getMilliseconds();
循环选择的每个元素 ②
    }
    var elements = document.getElementsByTagName('*'); ① 选择页面上的每个元素
    for(var i = 0; i < elements.length; i++) {
        (function(current) {
            current.onclick = function(event) {
                event = event || window.event;
                var target = event.target || event.srcElement;
                console.log(
                    'At ' + formatDate(new Date()) +
                    ' For ' + current.tagName + '#' + current.id +
                    ' target is ' + target.tagName + '#' + target.id
                );
            };
        })(elements[i]);
    }
</script>
</body>
</html>
每个元素定义一个 ③ onclick 处理器

```

对于这个例子，我们做了很多修改。首先删除mouseover事件之前的处理代码，这样可以只关注click事件。我们也把图片元素包装到三层div元素里，作为事件目标元素，故意把图片放在更深层次的DOM元素里。也可以给页面中的每个元素分配一个特殊的ID。

在<script>标签内部，JavaScript的getElementsByTagName()方法和通用选择器(universal selector)用于查找页面①上的所有元素。然后使用for循环来迭代每个元素②，而且让处理器响应点击事件③。对于每个匹配的元素，可以创建闭包(如果不熟悉闭包，可以阅读附录中有关闭包的内容)来记录当前局部变量里的实例。在处理器内部，不能直接引用elements[i]，因为此时处理器会被执行，索引i的值超出了数组对象索引的范围。

注意：getElementsByTagName()方法不返回实际的数组，而是返回HTMLCollection。像这样的对象，我们称为类数组(array-like)对象，因为它们允许使用索引来遍历对象内部的元素，而且包含length属性，但是并没有实现push()和join()方法。

该处理器采用了第6.1节中讨论的依赖于浏览器定位Event实例和辨别事件目标的技巧，然后在控制台上打印消息。这个消息是此例子中另外一个有趣的地方。它还显示标签的名字和当前元素的ID，让闭包工作，跟着目标的ID。基于此，每个控制台显示的消息都展示了当前元素事件冒泡的过程，也包括启动整个事件的目标元素。

加载例子页面(chapter-6/dom.0.propagation.html)，并点击图6.2中展示的图片结果。

这个例子清晰展示了当事件触发时，首先激活的是目标元素，然后是每一级父元素，直到html根节点元素。

这个功能十分强大，因为它允许我们在每一级元素上管理子元素的事件处理工作。思考一下form元素的处理器例子，它可以为每个子元素响应事件，基于元素新的动态值来显示页面。

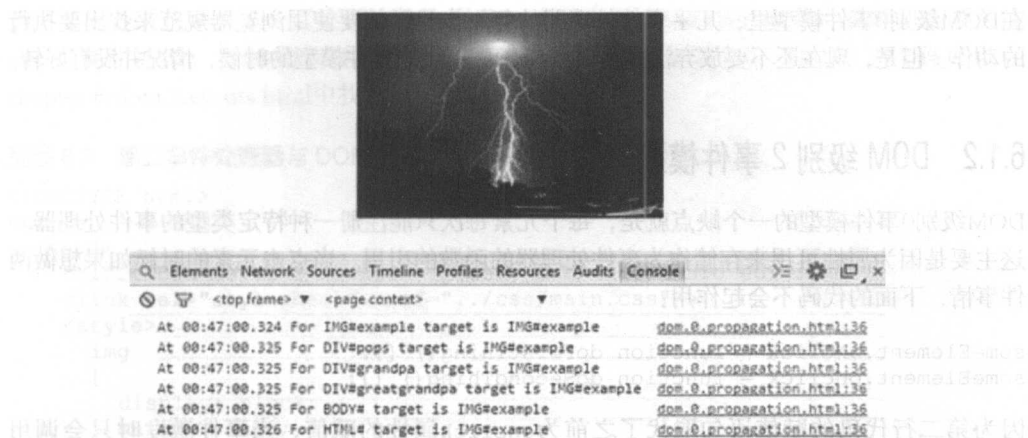


图 6.2 控制台消息清晰展示了事件冒泡的整个过程，从目标元素到 DOM 树的根节点

但是如果我们不希望事件传播呢？能够阻止它吗？

影响事件的传播和语义动作

许多场景希望阻止事件沿着 DOM 树向上冒泡传播。可能是因为我们挑剔，而且有对应的事件处理器，或者不希望高层节点出现意外的处理代码。

无论什么原因，都可以通过 Event 实例提供的机制来阻止事件向上传播。在现代浏览器中，可以调用 Event 实例的 `stopPropagation()` 方法来实现。在旧的 IE 浏览器中可以设置 Event 实例的 `cancelBubble` 属性为 `true` 来实现。比较有趣的是，许多新的兼容 W3C 标准的浏览器也提供了 `cancelBubble` 属性，尽管 `cancelBubble` 属性不属于 W3C 标准。

一些事件都有与它们相关的默认语义。例如，`anchor` 标签的 `click` 事件将会导致浏览器导航到元素的 `href` 属性，`form` 元素的 `submit` 事件将会导致表单提交。若想取消这些语义动作——通常作为默认的动作 (default action)——事件，在新的浏览器里可以调用 Event 实例的 `preventDefault()` 方法来阻止默认动作。在旧的 IE 里，这个方法不存在，所以要设置 `returnValue` 属性的值为 `false`。另外一种方法就是，在事件处理器代码里返回 `false`。有时候或许希望停止事件传播以及默认动作。

142

这样的动作通常出现在表单验证领域的时候。在表单提交 `submit` 事件的处理器代码里，可以编写代码来检查表单里控件的值，如果任何输入的数据错误，都可以返回 `false`。

也可能在 `form` 元素里见到代码：

```
<form name="myForm" onsubmit="return false;" ...
```

这行代码可以有效地阻止表单被提交，除了使用脚本控制(通过 `form.submit()`，不会触发 `submit` 事件)。

在DOM级别0事件模型里，几乎事件处理器的每一步代码都要使用浏览器规范来找出要执行的动作。但是，现在还不要放弃治疗——当考虑更高级的事件模型的时候，情况并没有好转。

6.1.2 DOM 级别 2 事件模型

DOM级别0事件模型的一个缺点就是，每个元素每次只能注册一种特定类型的事件处理器。这主要是因为属性可用来存储作为事件处理器的函数的引用。当点击元素的时候如果想做两件事情，下面的代码不会起作用：

```
someElement.onclick = function doFirstThing() {};  
someElement.onclick = function doSecondThing() {};
```

因为第二行代码的赋值语句取代了之前为onclick属性的赋值，当事件激发时只会调用doSecondThing()函数。当然，也可以把两个函数包装到第三个函数里，然后依次调用两个函数，但是作为页面来说，这会变得更加复杂，越来越难以跟踪这样的调用。此外，如果在页面里使用了多个可复用的组件或者库，它们可能不知道其他组件关于事件处理的需求。他人编写的代码也许正要为someElement.onclick属性赋值，这样代码可能就被重写了。也可以使用其他解决方案，但是所有的办法都会为页面带来更多的复杂性。

DOM级别2事件模型就是为了处理这种类型的问题而建立的。下面看看在这个高级模型下DOM元素如何建立多个事件处理器机制。

建立事件处理器

DOM级别2事件处理器通过标准的元素方法（method）而不是元素属性赋值函数来实现。每个DOM元素定义了一个名为addEventListener()的方法，它可以使用元素来添加事件侦听器。

此方法格式如下：

143 addEventListener(eventType, listener, useCapture)

eventType参数定义要处理的事件类型。这些字符串的值就是在DOM级别0事件模型中使用的事件，但是并不需要加on前缀，比如click、mouseover、keydown等。

listener参数表示作为元素事件处理器的函数（或者内联函数，通常是匿名函数）引用。在基本的事件模型里，Event实例传递给这个函数作为第一个参数。

最后的参数useCapture是个布尔值，我们会在后面讨论DOM级别2事件模型的事件传播时再介绍。现在把它设置为false即可。

现在已经讨论了addEventListener()的参数，猜一下接下来会介绍什么？旧版本的IE有自己的附加处理器的方式！我们会在第6.1.3节深入讨论这个问题。

为了实际查看这个方法，需要再次修改列表6.1来使用更高级的事件模型。这样只需要关注click事件类型。这时我们要为图片元素建立三个事件处理器。新的例子代码可以在chapter-6/dom.2.events.html中找到，如下：

列表 6.3 建立事件处理器与 DOM 级别 2 事件模型

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Level 2 Events - jQuery in Action, 3rd edition</title>
    <link rel="stylesheet" href="../../css/main.css"/>
    <style>
      img
      {
        display: block;
        margin: auto;
      }
    </style>
  </head>
  <body>
    
    <script>
      function formatDate(date) {
        return (date.getHours() < 10 ? '0' : '') + date.getHours() +
          ':' + (date.getMinutes() < 10 ? '0' : '') + date.getMinutes() +
          ':' + (date.getSeconds() < 10 ? '0' : '') + date.getSeconds() +
          '.' + (date.getMilliseconds() < 10 ?
            '00' : (date.getMilliseconds() < 100 ? '0' : '')) +
            date.getMilliseconds();
      }
      var element = document.getElementById('example');
      element.addEventListener('click', function(event) {
        console.log('At ' + formatDate(new Date()) + ' BOOM once!');
      }, false);
      element.addEventListener('click', function(event) {
        console.log('At ' + formatDate(new Date()) + ' BOOM twice!');
      }, false);
      element.addEventListener('click', function(event) {
        console.log('At ' + formatDate(new Date()) + ' BOOM thrice!');
      }, false);
    </script>
  </body>
</html>
```

144

img 元素建立三个事件处理器

这段代码非常简单，但是清晰地说明了如何为元素的同一个事件建立多个事件处理器——这在基本的事件模型（Basic Event Model）中无法轻易做到。在<script>标签内部，先获取图片元素的引用，然后为click事件建立三个事件处理器①。

在兼容标准的浏览器（除了IE8及以下）里加载这个页面，然后点击图片，如图6.3所示。

既然已经讨论了这个非常有用的特性，那么就要学习useCapture参数的作用。

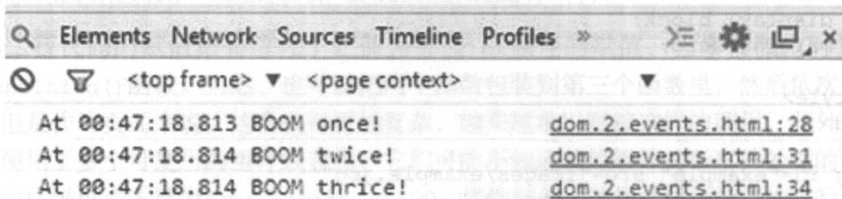
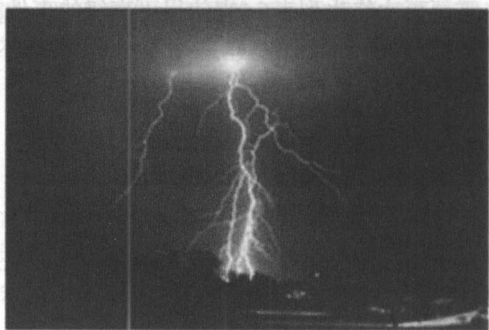


图 6.3 点击一次图片就会触发为 click 事件建立的三个事件处理器

事件传播

在DOM级别0事件模型里，一旦元素的某个事件被触发，事件就会从目标元素逐级向上传播直到DOM根节点。在高级的DOM级别2事件模型中，虽然提供了冒泡机制，但是也提供了额外的捕获阶段。

在DOM级别2事件模型中，当激发事件后，事件首先从DOM树根节点向目标元素传播，然后从目标元素向DOM根节点冒泡传播。前面的阶段(根到目标元素)被称为捕获阶段(capture phase)，而后者(目标到根)被称为冒泡阶段(bubble phase)。

当一个函数用来作为事件处理器时，它也可以被标记为捕获处理器，此阶段可以被触发，或者作为冒泡处理器被触发。可能你已经猜到addEventListener()方法的这个参数useCapture就是用来标记函数作为何种处理器使用的。false表示作为冒泡阶段处理器，true表示作为捕获处理器。这个参数是可选的，新版本的浏览器(Firefox 6开始)默认为false。

重新回头考虑列表6.2，再研究DOM层级中基本事件模型的传播过程。在那个例子中，我们在三层div里嵌入一个图片元素。在DOM级别2事件模型中，点击图片元素触发click事件，DOM树中的事件传播过程如图6.4所示。

让我们来测试这个过程吧。列表6.4展示了需要的页面代码(chapter-6/dom.2.propagation.html)。

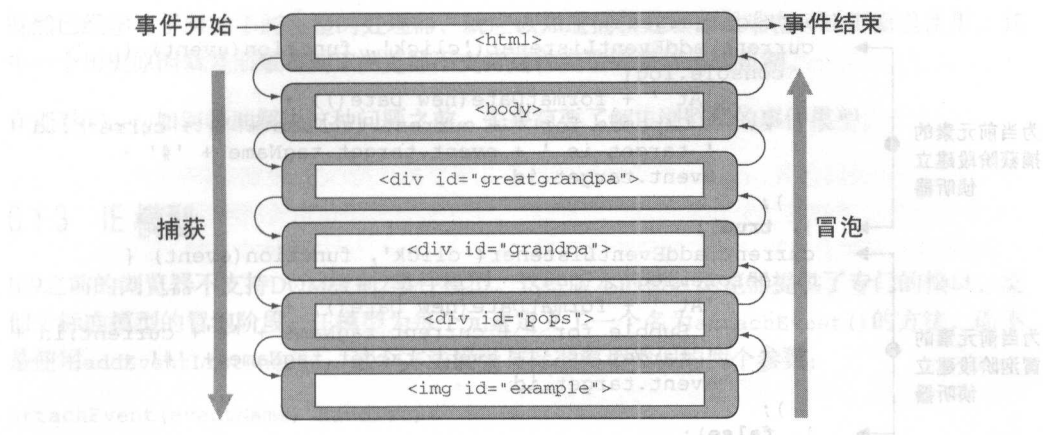


图 6.4 DOM 级别 2 事件模型中事件在 DOM 层级二次传播：一次从顶级到目标元素捕获阶段，一次从目标元素到根冒泡阶段

列表 6.4 追踪事件冒泡传播与捕获处理器

```

<!DOCTYPE html>
<html>
  <head>
    <title>DOM Level 2 Propagation - jQuery in Action, 3rd edition</title>
    <link rel="stylesheet" href="../css/main.css"/>
    <style>
      img
      {
        display: block;
        margin: auto;
      }
    </style>
  </head>
  <body>
    <div id="greatgrandpa">
      <div id="grandpa">
        <div id="pops">
          
        </div>
      </div>
    </div>
  </body>
</html>
<script>
  function formatDate(date) {
    return (date.getHours() < 10 ? '0' : '') + date.getHours() +
      ':' + (date.getMinutes() < 10 ? '0' : '') + date.getMinutes() +
      ':' + (date.getSeconds() < 10 ? '0' : '') + date.getSeconds() +
      '.' + (date.getMilliseconds() < 10 ?
        '00' : (date.getMilliseconds() < 100 ? '0' : '')) +
        date.getMilliseconds();
  }

  var elements = document.getElementsByTagName('*');
  for(var i = 0; i < elements.length; i++) {

```

```

    (function(current) {
        current.addEventListener('click', function(event) {
            console.log(
                'At ' + formatDate(new Date()) +
                ' Capture for ' + current.tagName + '#' + current.id +
                ' target is ' + event.target.tagName + '#' +
                event.target.id
            );
        }, true);
        current.addEventListener('click', function(event) {
            console.log(
                'At ' + formatDate(new Date()) +
                ' Bubble for ' + current.tagName + '#' + current.id +
                ' target is ' + event.target.tagName + '#' +
                event.target.id
            );
        }, false);
    })(elements[i]);
}
</script>
</body>
</html>

```

为当前元素的捕获阶段建立侦听器

为当前元素的冒泡阶段建立侦听器

为了建立多个事件处理器，这段代码修改了列表6.2来使用DOM级别2事件模型。在<script>标签中，使用getElementsByTagName()方法和通用选择器(universal selector)来查询页面元素。在每一个元素上都建立了两个处理器：一个捕获处理器①，一个冒泡处理器②。每个处理器都打印消息到控制台上以区分处理器类型、当前的元素及目标元素的ID。

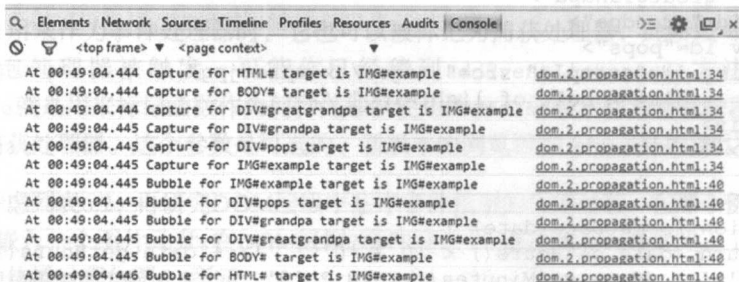


图 6.5 点击图片元素，会激发每个处理器在控制台打印消息，确认捕获阶段和冒泡阶段的事件处理过程

在Chrome浏览器里打开这个页面，点击图片，结果如图6.5所示，展示了事件处理两个阶段和DOM树的信息。注意：因为已经为目标元素定义了捕获处理器和冒泡处理器，两个处理器都会为目标元素和祖先元素一次执行。

既然已经学习了两种不同类型的处理器，就应该知道捕获处理器很难在Web页面里使用。其中一个历史原因就是旧版本的IE浏览器不支持这种类型的事件传播机制。

在查看jQuery如何帮助解决这种问题之前，先来简要了解IE浏览器的事件模型。

6.1.3 IE 模型

IE9之前的浏览器不支持DOM级别2事件模型。这些版本的微软浏览器提供了专门的接口，类似于标准模型的冒泡阶段。IE模型为每个元素定义了一个名为`attachEvent()`的方法，而不是使用`addEventListener()`。这个方法接受与标准模型类似的两个参数：

```
attachEvent(eventName, handler)
```

第一个参数接受事件类型，但是使用了与DOM级别0事件模型一样的名字，比如`onclick`、`onmouseover`、`onkeydown`等。

148

第二个参数是用来作为事件处理器的函数名称，而且Event实例必须从`window.event`属性获取。此外，这个方法不支持任意的捕获阶段。

尽管使用了相对独立的DOM级别0事件模型，但还是要面对各个事件处理阶段许多浏览器独立的选择问题。而且当使用DOM级别2或者IE模型时，为了支持更广泛的用户，还必须建立不同的事件处理器代码。

jQuery提供了对浏览器差异性的封装，以简化开发者的工作。

我们来看看如何实现吧！

6.2 jQuery 事件模型

The jQuery Event Model

虽然创建高交互性的网络应用需要高度依赖事件处理代码，但是需要开发者编写大型网站事件处理代码并且处理浏览器之间的差异性，这种需求会吓着许多网页开发者。

可以把这些差异隐藏到API后，这些API已经从页面代码中抽象封装了差异。但是为什么不适用jQuery封装好的代码呢？

jQuery的事件模型实现代码，我们称为jQuery事件模型(jQuery Event Model)，具备以下特点：

- 为建立事件处理器提供统一方法。
- 允许每个元素、每个事件类型注册多个方法。
- 适用标准事件名称，如`click` 或 `mouseover`。

- 传递Event实例作为第一个参数。
- 规范事件实例中最常用的属性。
- 为事件取消和阻止操作提供统一的方法。

除了不支持捕获阶段外，jQuery事件模型的特性几乎完美支持DOM级别2事件模型，使用单一的API支持标准兼容的浏览器和旧的IE浏览器。对于大多数从不使用捕获阶段(甚至不知道它的存在)的网页开发者来说，忽略捕获阶段不是问题，因为在旧的IE里缺少支持。但是真的这么简单吗？下面来看看究竟。

6.2.1 使用 jQuery 附加事件处理器

使用jQuery事件模型，可以使用`on()`方法给DOM元素添加事件处理器。目前为止，已经看过如何给元素添加一个或者多个事件处理器，但是jQuery的一大好处就是可以使用强大的`jQuery()`方法来查询元素集合，并且为所有的元素在一行代码里添加相同的处理器。思考下面的简单例子：

```
149 > $('img').on('click', function(event) {  
    alert('Hi there!');  
});
```

这段代码为页面上的所有图片的`click`事件添加了内联匿名函数作为事件处理器。`on()`方法的说明如下。

on()方法语法

`on(eventType[, selector][, data], handler)`

`on(eventsHash[, selector][, data])`

为选择的元素的一个或者多个事件添加一个或者多个事件处理器

参数

`eventType(String)` 指定要建立处理器的事件类型的名称(表6.1里有完整的列表)。多个事件类型可以通过用空格隔开来指定。

这些事件类型也可以添加命名空间，使用圆点后加上命名空间就可(例如`click.myapp`)。允许有多个命名空间(例如，`click.myapp.mymodule`)。

`selector(String)` 可选参数，过滤器用来过滤选中元素的子元素，这些子元素触发事件。如果省略选择器，当到达被选中的元素时就会触发事件。

`data(Any)` 传递给Event实例的数据，为`data`属性赋值，事件处理器函数可以访问。

`handler(Function)` 作为事件处理器的函数，当调用的时候，Event实例作为第一个参数，冒泡阶段的当前元素作为函数调用上下文(`this`)。`false` 值可以表示函数`return false`，函数可以通过使用`trigger()`或者`triggerHandler()`接受其他参数(本章后面讨论)。

eventsHash(Object) 在单个调用中为多个事件类型建立处理器的JavaScript对象。属性名区分事件类型(与使用eventType参数一样), 而且属性值提供事件处理器。

返回

jQuery集合。

jQuery 3:签名改进

jQuery 3允许处理器参数为对象。虽在编写本书时还没有太多的改进细节, 但是不要担心: 这并不常用。如果想学习更多关于这个特性的知识, 可以查看GitHub:<https://github.com/jquery/jquery/issues/1735>。

在深入介绍这个重要的jQuery方法之前, 我们看一个基本的实战例子。列表6.3的代码从DOM级别2事件模型转化为jQuery事件模型, 新的代码如列表6.5所示, 源代码在chapter-6/jquery.events.html中。

150

列表 6.5 无浏览器特定代码来建立高级事件处理器

```
<!DOCTYPE html>
<html>
  <head>
    <title>Events in jQuery Example - jQuery in Action, 3rd edition</title>
    <link rel="stylesheet" href="../css/main.css"/>
    <style>
      img
      {
        display: block;
        margin: auto;
      }
    </style>
  </head>
  <body>
    

    <script src="../js/jquery-1.11.3.min.js"></script>
    <script>
      function formatDate(date) {
        return (date.getHours() < 10 ? '0' : '') + date.getHours() +
          ':' + (date.getMinutes() < 10 ? '0' : '') + date.getMinutes() +
          ':' + (date.getSeconds() < 10 ? '0' : '') + date.getSeconds() +
          '.' + (date.getMilliseconds() < 10 ?
            '00' : (date.getMilliseconds() < 100 ? '0' : '')) +
            date.getMilliseconds();
      }
    </script>
```

```

$('#example')
    .on('click', function (event) {
        console.log('At ' + formatDate(new Date()) + ' BOOM once!');
    })
    .on('click', function (event) {
        console.log('At ' + formatDate(new Date()) + ' BOOM twice!');
    })
    .on('click', function (event) {
        console.log('At ' + formatDate(new Date()) + ' BOOM thrice!');
    });
</script>
</body>
</html>

```

使用 jQuery 绑定事件处理器给图片

代码的修改仅限于如何附加事件处理器❶，虽简单但非常重要。我们创建了一个目标元素组成的集合，在这个元素上调用此方法on()——记住，jQuery链式调用允许在单个语句里添加多个方法——每个方法都会作为事件处理器。

在支持jQuery的浏览器里加载页面(可以最终忘记兼容标准的浏览器问题了)，然后点击图片，显示结果如图6.6所示，不出意外，与图6.3所示的结果完全相同。

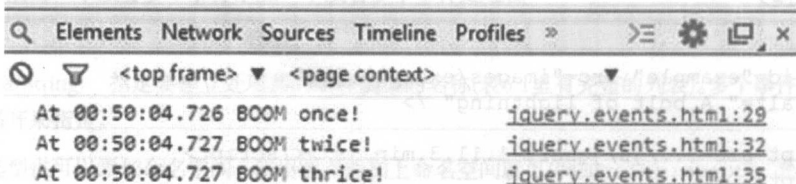
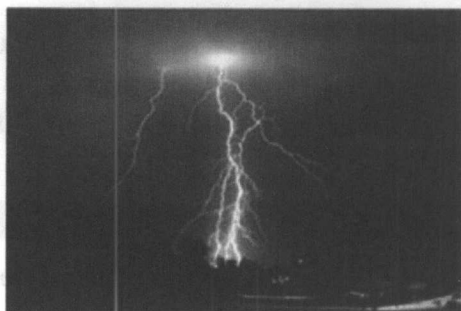


图 6.6 jQuery 事件模型允许指定多个事件处理器，就像 DOM 级别 2 事件模型

这段代码也可以在旧的IE浏览器上工作(取决于jQuery版本)，不能在不添加专门浏览器测试代码的时候使用列表6.3中的代码，比如针对当前的浏览器使用正确的代码。

值得注意的是，与其他的jQuery方法不同，selector参数并不是在on()方法调用的时候去进一步过滤jQuery集合中的对象来接受事件处理器的。它在事件发生时用来确定事件处理器是否被调用。这个概念会在后面澄清，现在会看到一些具体的例子，稍作等待。

jQuery 3:方法过时

`on()` 方法提供了统一的接口来取代jQuery的`bind()`、`delegate()`和`live()`方法。`live()`在1.7版本中过时，在1.9版本中删除，虽然`bind()`和`delegate()`还在，但是强烈不建议使用。jQuery 3不赞成使用`bind()`、`delegate()`方法，所以建议使用`on()`方法（这也是为什么在本书里没有使用这个旧方法的原因）。

`on()` 方法与原生的添加事件处理器方法有很大不同。使用jQuery添加的事件处理器返回`false`，与调用`preventDefault()`和`stopPropagation()`一样；而使用原生方法添加的事件处理器返回`false`，与调用`preventDefault()`一样。

此刻，那些纠结于浏览器差异事件处理器代码的网页开发者正在欢唱“幸福的日子又来了（Happy Days Are Here Again）”，在座椅中转起来。谁又能责怪他们呢？

列表6.5展示了`on()`方法多么灵活和聪明。如果只想传递处理器而不包括`data`或`selector`，那么不需要像这样传递`null`参数。`on()`方法允许传递处理器作为第二个参数。代替下面的写法： ◀ 152

```
$('#img').on('click', null, null, function() { ... });
```

可以这样编写代码：

```
$('#img').on('click', function() { ... });
```

解释这些结果已经超出本书的范畴，但是强烈推荐你阅读源码来找出答案。

在`on()`方法的介绍中，我们知道第一个参数可以是包含多个以空格分割事件的列表。使用这个变量的例子如下：

```
$('#button').on('click', function(event) {  
    console.log('Button clicked!');  
})  
.on('mouseenter mouseleave', myFunctionHandler);
```

在这段代码中，查询所有的`<button>`，然后附加上三个事件处理器。第一个是匿名函数，触发`click`事件时执行。第二个是`myFunctionHandler`，假想的函数在别处声明，触发`mouseenter`或者`mouseleave`时执行。

在`on()`签名介绍中，也强调了第一个参数可以是名为`eventsHash`的JavaScript对象，它的属性用来区分事件类型，属性值提供事件处理器。使用这个方法，之前的代码可以被重写为

```
$('#button').on({  
    click: function(event) {  
        console.log('Button clicked!');  
    },  
    mouseenter: myFunctionHandler,  
    mouseleave: myFunctionHandler  
});
```

选择哪个版本取决于你，我们建议选择并坚持一种风格。

有时需要给事件处理器传递数据。虽然可以在变量里存储数据，然后依赖于闭包，但有些简单的情况可以使用data参数。假设有一个处理器设置给了button的click事件，它会打印人的名字到控制台上。可以使用data参数来传递name，代码如下：

```
$('#my-button').on('click', {
  name: 'John Resig'
}, function (event) {
  console.log('The name is: ' + event.data.name);
});
```

正如这段代码所示，可以通过Event实例(event)的data属性来获取对象的name属性。如果想看现场演示，那么可以在chapter-6/on.data.parameter.html和JS Bin(<http://jsbin.com/IVONuWol/edit?html,js,console,output>)中查看。

目前为止，已经为页面HTML标签中的DOM元素附加了事件处理器。但是这些元素如果不存在怎么办？

正如之前讨论的，jQuery允许通过添加、修改或者删除元素来动态地操作DOM元素。当使用Ajax技术以后，很可能页面生命周期内的DOM元素会频繁出现或者消失，特别是在处理单页应用程序的时候。

解决这个问题的办法称为事件委托(event delegation)。事件委托是一种向元素的父元素添加事件处理器的重要技术。事件委托可以用在原始的JavaScript代码中，但是这只是jQuery的强大特性之一。假设有一个无序的列表，已经使用Ajax被五个元素填充。此时页面的脚本要执行，列表仍然为空：

```
<ul id="my-list"></ul>
```

在Ajax调用后，列表元素如下：

```
<ul id="my-list">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
  <li>Item 4</li>
  <li>Item 5</li>
</ul>
```

当鼠标悬浮在列表上时，要在控制台打印出元素的索引。现在对比如何通过原生的JavaScript和jQuery来实现事件委托。

使用原生的JavaScript，代码如下：

```
document.getElementById('my-list').addEventListener('mouseover',
  function(event) {
    if (event.target.nodeName === 'LI') {
      console.log('List item: ' +
```

```

        (Array.prototype.indexOf.call(
            document.getElementById('my-list').children,
            event.target
        ) + 1)
    );
}
},
false
);

```

154

使用jQuery代码可以减少到3行(使用相同的代码风格比较公平)，如下：

```

$('#my-list').on('mouseover', 'li', function(event) {
    console.log('List item: ' + ($(this).index() + 1));
});

```

这是干净的代码！你感觉只是获取了更少的代码吗？不是！你认为原生的JavaScript版本不支持旧的IE而jQuery支持吗？当然，在其背后jQuery执行了相同的操作，但是，当jQuery执行工作的时候，为什么还要担心兼容性问题？此刻，你应该开始明白为什么jQuery被如此广泛地使用。如果你想练习这些代码，那么可以在chapter-6/javascript.event.delegation.html和JS Bin(<http://jsbin.com/fihixa/edit?html,js,console,output>)中找到。展示jQuery如何实现事件委托的代码可以在chapter-6/jquery.event.delegation.html和JS Bin(<http://jsbin.com/bobaza/edit?html,js,console,output>)中找到。

事件委托的优势并非只局限于为不存在的元素执行处理器。它还可以节约内存和时间。假设例子中Ajax调用使用几百个项目来填充列表而不是五个。为了直接给列表添加元素，可以循环遍历每个项目(可以使用jQuery()方法，背后也可以迭代每个元素)。这个过程需要一定的时间，或许几百毫秒。当执行循环的时候，浏览器的用户界面被阻塞，而且用户感知性能差。此外，由于JavaScript是一门单线程语言，因此当循环执行的时候没有办法执行其他操作。很明显，附加处理器给一个元素需要的时间更少。我们说过，事件委托可以节约内存。在这个例子里，不必为一个元素保留几百个处理器。这样节约了大量的内存！当实现事件委托时，可以看到附加给document的代码。修改这个例子，jQuery代码可以重新组织如下：

```

$(document).on('mouseover', '#my-list li', function(event) {
    console.log('List item: ' + ($(this).index() + 1));
});

```

注意，已经更新了selector参数来确保可以在相同的集合上附加事件处理器。因为这个例子，你也许会认为，把所有的处理器附加给document是一个好主意，但是为DOM根元素或者document添加太多的处理器会降低性能。在这个例子中，每次触发事件，jQuery都必须为从事件目标元素到顶部(因为事件冒泡)每个元素比较提供的selector参数和触发事件的元素。为了避免这个问题，更好的做法是，尽可能为离目标近的元素添加委托事件。

155

到此，已经讨论了很多关于事件的内容，也使用了其中的一些事件。但是，还有很多其他的事件，这些事件是做什么的？表6.1列出了可以侦听的事件。

表 6.1 侦听可用的事件

事 件			
blur	focusin	mousedown	mouseup
change	focusout	mouseenter	ready
click	keydown	mouseleave	resize
dblclick	keypress	mousemove	scroll
error	keyup	mouseout	select
focus	load	mouseover	submit
			unload

既然已经学习了所有关于事件处理器的知识，现在讨论一下on()方法的变种，允许附加一次处理器，使用一次，然后删除。

侦听一个事件一次

jQuery提供特殊的on()方法，名字叫one()，它建立的处理器就是一锤子买卖。一旦事件处理器执行，就会自动删除。其语法与on()的一样，所以不再重复解释(如果需要复习，可以看一下on()方法的语法)。

one()方法语法

one(eventType[, selector][, data], handler)

one(eventsHash[, selector][, data])

为匹配集合中的所有元素的特定事件建立事件处理器。一旦执行，处理器就会自动删除。

返回

jQuery集合。

到目前为止，已经知道如何为集合元素添加事件处理器，但是可能最终还要删除这些处理器。下面看看如何实现。

6.2.2 删除事件处理器

通常情况下，一旦使用on()方法建立事件处理器，它就会在整个页面生命周期里起作用。但是有些情况下我们希望根据条件来删除事件处理器。思考一下，例如，一个页面有多个展示步骤，一旦一个步骤完成，它的控件就会变成只读。

对于这种情况，使用脚本删除事件处理器可以节约内存。我们已经介绍了one()方法在执行完处理器后自动删除，但是对于更多的通用情况，要想自己控制删除的操作时机，jQuery提供了on()对应的方法off()。它的语法如下所述，与on()和one()参数相同，所以不再重复介绍。


```
off(eventType[, selector][, handler])
```

```
off(eventsHash, [, selector])
```

```
off()
```

删除参数指定的jQuery对象中所有元素的事件处理器。如果没有提供参数，则会删除所有的元素处理器。

返回

jQuery集合。

可以使用该方法在不同的粒度级别删除jQuery对象中元素的事件处理器。忽略参数可以删除所有的事件侦听器。删除特定类型的事件侦听器只需要提供事件类型参数。如果提供一个或者多个类型名字，那么所有的事件被删除，委托不一定删除。最后，特定的处理器可以通过提供最初处理器函数的引用来删除。要实现这个目标，必须在绑定函数处理器的时候保留对函数的引用。因此，当删除最初作为侦听器的函数时，它或者被定义为顶级函数（这样可以被顶级变量引用），或者以其他方式保存。内联函数无法被后续off()调用。

jQuery 3:方法过时

off()方法提供统一的接口来替换unbind()、undelegated()和die()(多么恐怖的名字)方法。就像它们各自的对应方法，die()方法(对应于live()方法)在1.7版本中过时，在1.9版本中被删除，而unbind()方法(与bind()对应)和undelegate()方法(与delegate()方法对应)仍然在核心库里，但是不鼓励使用它们。unbind()和undelegate()在jQuery 3中都已经过时，建议使用off()方法。

对于匿名内联函数，使用命名空间化的事件十分方便，因为可以在特定的命名空间内解除所有事件的绑定，但保留个别侦听器的引用。例如，

```
$('#*').off('.fred');
```

157

将会删除所有fred命名空间下的事件侦听器(记住这种情况必须要求fred不是样式选择器)。这个用法对于从jQuery插件中附加事件处理器非常有用，将会在第12章详细讨论。

在继续下面的内容之前，先来看下使用on()和off()的例子。思考下面的代码，假设有三个按钮：

```
<button id="btn">Does nothing</button>
<button id="btn-attach">Attach handler</button>
<button id="btn-remove">Remove handler</button>
```

此外，还有下面的代码：

```
var $btn = $('#btn');
var counter = 1;
```

```
function logHandler() {
```

① 定义一个在控制台显示执行次数的函数

```

    console.log('click ' + counter);
    counter++;
  };

  $('#btn-attach').on('click', function() {
    $btn
      .on('click', logHandler)
      .text('Log');
  });

  $('#btn-remove').on('click', function() {
    $btn
      .off('click', logHandler)
      .text('Does nothing');
  });

```

② 为 btn-attach 按钮添加点击处理器

③ 为 btn-remove 按钮添加处理器

想法是：第一个按钮自己不做工作；其他两个按钮，分别为第一个按钮附加事件处理器，然后删除处理器。在处理器内部，当事件处理器附加给按钮时①，在控制台上打印按钮点击的次数。

为此，在代码的第一部分可以声明一个变量\$btn，包含一个元素的集合、第一个按钮元素（ID为btn）。然后定义计数器(counter)以及作为侦听器的函数。在代码的第二部分，可以为第二个按钮（ID为btn-attach）附加内联函数作为处理器，使用的是on()方法②。它的作用就是为第一个按钮添加logHandler作为事件处理器。同样，可以使用内联函数为第三个按钮（ID为btn-remove）添加处理器③，点击时为第一个按钮删除事件处理器。可以在chapter-6/adding.removing.handlers.html以及JS Bin(<http://jsbin.com/iqurujug/edit?html,js,console,output>)查看例子。注意：多次点击添加按钮会为第一个按钮添加多个同样的处理器，所以计数器会根据Log按钮点击次数的增加而增加。

目前为止，已经介绍了jQuery事件模型，它可以非常方便地建立(删除)事件处理器，而不需要担心浏览器兼容性问题，但是，如果自己编写事件处理器代码呢？

6.2.3 监测事件实例

当使用on()方法或者其他方法建立事件处理器时，Event实例会作为第一个参数传递给函数，无论什么浏览器，无须担心旧的IE浏览器的window.event属性。但是，仍然需要处理Event实例的不同属性，不是吗？

值得庆幸的是，不需要，因为jQuery实际上并没有传递Event实例给事件处理器。

是的，我们一直在隐藏这个小细节，因为到目前为止，它还不是问题。但是现在，已经进入更高阶段，可以学习处理器内部的事件实例了，马上就会真相大白！

事实上，jQuery提供了jQuery.Event对象来传递给处理器。但是请原谅我们的简化，因为jQuery复制了Event绝大部分属性给这个对象。隐藏，如果需要查找Event的属性，这个对象几乎与最初的Event对象实例没有区别。

但是，这并非此对象最重要的方面。真正有价值的，是这个对象存在，包含一系列标准化的值和方法，可以独立于浏览器使用，忽略Event对象的差别。

表6.2列举了可以独立访问的jQuery.Event属性。注意一些属性可能包含undefined值，这取决于事件触发。

表 6.2 浏览器独立的 jQuery.Event 属性

		属 性	
altKey	currentTarget	offsetY	screenX
bubbles	data	originalTarget	screenY
button	detail	originalEvent	shiftKey
cancelable	delegateTarget	pageX*	target*
charCode	eventPhase	pageY*	timestamp
clientX	metaKey*	prevValue	type
clientY	namespace	relatedTarget*	view
ctrlKey	offsetX	result	which*

正如你看到的，有些属性使用了星号标记。原因是jQuery为跨浏览器兼容性标准化了这些属性。意思是它们在一些浏览器中命名不同（旧的IE浏览器）。为了避免记住所有差别带来的痛苦，jQuery提供了一个属性名，而且内部处理了这些差异性。其中一个例子就是target属性，它在旧的IE浏览器里叫srcElement。此外，一些事件或许有专门的属性，可以通过originalEvent属性访问。

jQuery.Event对象也包含几个方法，如表6.3所示。

表 6.3 浏览器独立的 jQuery.Event 方法

方 法	
preventDefault()	阻止任意默认的语义动作发生(比如表单提交、超链接跳转、勾选框状态改变等)
stopPropagation()	阻止事件沿着 DOM 树向上传播,当前目标的其他事件不受影响,支持使用浏览器事件和自定义事件
stopImmediatePropagation()	阻止所有的事件传播,包括当前目标的其他事件
isDefaultPrevented()	如果在这个实例上调用 preventDefault() 方法,就会返回 true
isPropagationStopped()	如果在这个实例上调用 stopPropagation() 方法,就会返回 true
isImmediatePropagationStopped()	如果在这个实例上调用 stopImmediatePropagation() 方法,就会返回 true

除了允许以浏览器独立的方式来管理事件处理以及Event对象外，jQuery还提供使用脚本控制的触发事件和运行事件的方法集合。下面看看这些方法。

6.2.4 触发事件

事件处理器设计用来在浏览器或者用户行为在触发DOM树层级中的特定事件时调用。但是有时候，系统用脚本控制事件的触发和执行过程。也可以定义这样的事件处理器，作为顶级函数，这样就可以通过名字来调用它们，正如我们看到的，作为内联匿名函数定义事件处理器非常常见，而且相当方便！此外，调用作为处理器的函数不会导致语义动作或者冒泡传播发生。

160 为此，jQuery提供了脚本专用的触发事件方法。最通用的就是`trigger()`方法，它的语法如下。

trigger()方法语法

trigger(eventType[, data])

为所有匹配的元素调用传递事件类型建立任意事件处理器和行为。

参数

`event(String|jQuery.Event)` 指定要调用的事件类型的名字，也可以包含命名空间。另外，也可以传递`jQuery.Event`。

`data(Any)` 传递给处理器的数据。如果是数组，传递的元素会作为不同的参数。

返回

jQuery集合。

`trigger()`方法，包括将要介绍的方便的方法，尽全力模拟事件被触发，包括DOM层级传播和语义动作执行。

每个处理器都会在调用时传递`jQuery.Event`的实例。因为没有真正地报告特定事件值的事件和属性，比如鼠标事件的位置或者键盘事件的键，所以不会被传递（属性根本不存在，这与它们的值是`undefined`不同）。`target`属性用于设置匹配集合元素的引用、事件处理器绑定的目标。

正如真实的事件，引发事件的传播可以通过调用该`jQuery.Event`实例的`stopPropagation()`方法叫停，或者从任意处理器中返回`false`。

传递给`trigger()`方法的`data`参数与建立事件处理器时传递的参数不同。后者被放进`jQuery.Event`实例的`data`属性；传递给`trigger()`的值（正如将要看到的`triggerHandler()`）作为参数传递给侦听器。这允许两个数据值都可以使用而不会互相冲突。

在深入学习`triggerHandler()`之前，先来详细讨论`trigger()`方法的`data`参数，以及传递数组与其他数据类型有什么不同。思考下面的代码块，它为ID是`foo`的元素添加一个事件处理器，当触发`click`事件的时候执行。

```
$('#foo').on('click', function(event, par1, par2, par3){
    console.log(par1, par2, par3);
});
```

正如我们看到的，定义的event参数也有三个新的参数：par1、par2及par3。在处理器内部打印三个参数的值。现在思考使用jQuery的trigger()方法来执行处理器，传递三个数字(1, 2, 3)来促发事件：

```
$('#foo').trigger('click', 1, 2, 3);
```

这样，就可以在控制台上看到如下结果：

```
1 undefined undefined
```

发生这个错误，主要是触发器方法只接受一个参数传递给处理器，所以其他的(2和3)都被忽略了。如果想提供多个参数，则可以使用如下所示的数组方法：

```
$('#foo').trigger('click', [1, 2, 3]);
```

一旦附加的元素处理器执行，下面的结果就会打印到控制台上，数组中的元素作为独立的参数传递（par2和par3不会再是undefined）：

```
1 2 3
```

如果想要做更多关于data的练习，则可以在chapter-6/trigger.data.parameter.html以及JS Bin(<http://jsbin.com/mefohu/edit?html,js,console,output>)中找到例子。

对于想要触发处理器但是不会导致事件传播和执行语义动作的情况，jQuery提供了triggerHandler()方法，它与trigger()方法类似，不同的是没有冒泡传播或者语义动作。

triggerHandler()方法语法

triggerHandler(eventType[, data])

为匹配的元素调用建立的事件处理器，没有出现冒泡传播、语义动作或者实时事件。此方法返回最后一个处理器的返回值，或者undefined。

参数

eventTpye(String) 指定要激活事件类型的名字。

data(Any) 传递给处理器的数据，如果设置数组，则元素会作为不同的参数传递。

返回

如果触发器或者处理器没有返回值，则返回undefined。

triggerHandler()方法与trigger()方法一样都包含data参数，因此之前的演示对于triggerHandler()方法一样适用，但是它们还是有些重要的不同点值得讨论。首先，trigger()方法会在jQuery集合中的所有元素上执行，triggerHandler()方法会在第一个元素上执行。此外，triggerHandler()方法不允许链式调用，因为它的返回值与处理器返回值一样(如果处理器没有返回值，则会返回undefined)，而trigger()方法返回匹配的元素。最后，triggerHandler()方法不会在DOM层级中冒泡传播。

没有文字可以代替好的例子，所以在介绍下面的内容之前，来看看trigger()和triggerHandler()的实战例子（这本书不是叫“jQuery实战”吗）。下面的例子可以在chapter-6/jquery.triggering.events.html和JS Bin(http://jsbin.com/AqaqAGO/edit?html,css,js,console,output)中找到。

列表 6.6 jQuery 中的触发事件

```
<!DOCTYPE html>
<html>
  <head>
    <title>Triggering Events - jQuery in Action, 3rd edition</title>
    <link rel="stylesheet" href="../../css/main.css"/>
    <style>
      #wrapper
      {
        border: 1px solid #3A5895;
        padding: 10px;
      }

      #address:focus
      {
        border: 3px solid #000000;
      }
    </style>
  </head>
  <body>
    <div id="wrapper">
      <button id="btn">Click me!</button>
      <input type="text" id="address" />
    </div>

    <script src="../../js/jquery-1.11.3.min.js"></script>
    <script>
      $('#wrapper')
        .on('focus', function() {
          console.log('Div focused');
        })
        .on('click', function() {
          console.log('Div clicked');
        });
      $('#address')
        .on('focus', function() {
          console.log('Input focused');
        })
        .triggerHandler('focus');
      $('#btn')
        .on('click', function() {
          console.log('Button clicked');
        })
        .trigger('click');
    </script>
  </body>
</html>
```

为 div 的 focus 和 click 事件添加两个处理器

为 input 元素添加 focus 处理器

使用 triggerHandler() 触发 input 元素的 input 事件

为按钮添加处理器，当点击时执行

使用 trigger() 触发按钮点击事件

列表6.6中的代码可以让你看到trigger()和trigger-handler()行为之前的讨论。打开页面，

应该看到一个布局，如图6.7所示。

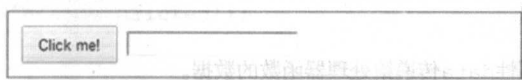


图 6.7 页面布局 jquery.triggering.events.html

在列表6.6中创建了一个`<div>`元素来包围其他两个元素：一个按钮和一个输入框。在script元素里，首先选择包围元素，然后使用`on()`方法附加两个侦听器：一个是聚焦时，一个是点击时❶。在内部代码里，在控制台打印什么元素和什么事情被调用。然后为Id是address的输入框元素添加聚焦处理器❷。感谢链式调用，在结束代码分号之前，也可以调用`triggerHandler()`方法，传递字符串作为参数❸通过调用`triggerHandler()`，可以执行附加的函数。正如我们说的，`triggerHandler()`不会冒泡传播，所以附加给ID为wrapper元素的处理器不会执行。

最后，选择按钮元素，为click事件添加处理器❹。对于输入框元素，在结束代码前，可以触发事件，但是这次试用的是`trigger()`方法❺。因此，在控制台输出消息后，事件冒泡传播，所以附加给div的点击事件的处理器也被执行了。

基于这个例子的讨论，只要加载页面，就会看到如图6.8所示的结果。

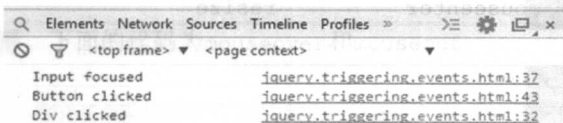


图 6.8 jquery.triggering.events.html 页面的输出结果

像`on()`和`trigger()`这样的方法会经常用到，所以每次编写这种完整语法的代码会非常烦人。jQuery团队完全能体会这种痛苦，所以他们引入了一系列的快捷方式。

164

6.2.5 快捷方式

jQuery提供了大量的快捷方式来建立特定类型的事件处理器以及触发事件。因为这些方法除了方法名，其他语法都是相同的。为节约空间，将它们列举到下面的单一方法描述里。

指定事件绑定方法语法

`eventName([data,] handler)`

通过方法名为指定事件使用的函数建立事件处理器，支持的方法如下：

blur	focusout	mouseleave	resize
change	keydown	mousemove	scroll
click	keypress	mouseout	select
dblclick	keyup	mouseover	submit
focus	mousedown	mouseup	
focusin	mouseenter	ready	

指定事件绑定方法语法

参数

`data(Any)` 作为Event属性data传递给处理器函数的数据。

`handler(Function)` 作为事件处理器的函数。

返回

jQuery集合。

除了添加事件处理器函数，这些方法还可以像`trigger()`一样使用。所有这些方法的语法除了方法名外，其他都类似，语法如下。

指定事件触发方法语法

`eventName()`

为所有匹配的元素调用指定名称的任意事件处理器和行为，支持的方法如下：

<code>blur</code>	<code>focusout</code>	<code>mouseleave</code>	<code>scroll</code>
<code>change</code>	<code>keydown</code>	<code>mousemove</code>	<code>select</code>
<code>click</code>	<code>keypress</code>	<code>mouseout</code>	<code>submit</code>
<code>dblclick</code>	<code>keyup</code>	<code>mouseover</code>	
<code>focus</code>	<code>mousedown</code>	<code>mouseup</code>	
<code>focusin</code>	<code>mouseenter</code>	<code>resize</code>	

参数

无。

返回

165 > jQuery集合。

jQuery 3:删除的方法

jQuery 3删除了已经过时的`load()`、`unload()`、`error()`方法。这些方法没有列举在前面的介绍里，因为它们很早就过时了(自jQuery 1.8)。如果在项目里仍然使用它们或者是依赖的插件，那么升级到jQuery 3会导致代码出错。

为了方便大家了解什么是快捷方式，我们修改列表6.6的代码，这样就可以使用快捷方式了。为了便于大家阅读，我们把代码贴到下面：

```
$('#btn')
  .on('click', function() {
    console.log('Button clicked');
  })
  .trigger('click');
```

使用快捷方式后，代码可以简化如下：

```

$('#btn')
    .click(function() {
        console.log('Button clicked');
    })
    .click();

```

除了这些快捷方式，jQuery里还有一个不同的地方。

悬浮在元素上

交互性网站里经常使用的多事件场景就是包含鼠标的进入和退出元素。通知什么时候进入和离开元素的事件，对于建立展示给用户的用户界面元素来说至关重要。这些元素类型中，作为导航的级联菜单就是一个常见的例子。

mouseover和mouseout事件类型的复杂行为常常阻碍我们创建这样的元素：

mouseout事件激发就是当鼠标移动到某个区域元素的外部。思考图6.9展示的布局，可在chapter-6/hover.html和JS Bin(<http://jsbin.com/nobuti/edit?html,js,console,output>)中找到这个代码。

这个页面展示了两个类似的区域集合(除了名字)：外部区域和内部区域。在浏览器里打开这个页面，按照下面的步骤操作。

对于页面顶部的矩形，下面的代码为mouseover和mouseout事件建立处理器：

```

$('#outer1').on('mouseover mouseout', report);
$('#inner1').on('mouseover mouseout', report);

```

这些代码使用名为report的函数为mouseover和mouseout建立处理器，report函数的定义如下：

```

function report(event) {
    event.stopPropagation();
    console.log(event.type + ' on ' + event.target.id);
}

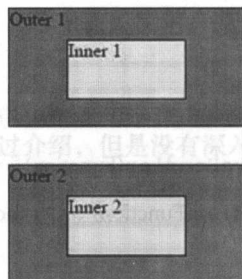
```

这个侦听器首先阻止事件冒泡传播，然后在控制台打印一些包含事件名称和激发事件的元素ID的文字信息。

现在移动鼠标指针进入“Outer1”的区域，会看到控制台上mouseover事件被触发。把鼠标指针移出区域，正如所料，会看到mouseout事件被触发。

现在移动鼠标指针进入“Outer1”，但是这次继续移动直到鼠标指针进入“Inner1”。mouseover事件被激活，mouseout事件被“Outer1”激活。如果在“Outer1”和“Inner1”来回挥动鼠标，就会看到一系列mouseover事件。这是定义的行为，尽管还不够直观。

尽管鼠标指针还在“Outer1”的边框内，当指针进入内部元素时，事件模型会认为它要离开外



166

图 6.9 这个页面可帮助演示当鼠标的指针移动到区域和子区域之上时，触发鼠标事件

部区域进入内部区域。

预期与否，我们不会一直想要这样的行为。通常希望在指针离开区域的时候被通知，而不关注它是否进入区域内。

幸运的是，大部分浏览器都支持`mouseenter`和`mouseleave`事件，首先在IE浏览器引入。这个事件对一起使用会更直观，从元素移动到子元素不会促发`mouseleave`事件。

使用jQuery可以为一系列事件建立处理器，代码如下：

```
$(element).mouseenter(function1).mouseleave(function2);
```

167➤ 但是，jQuery也提供了更加单一的方法来简化开发：`hover()`。此方法的语法描述如下。

hover()方法语法

hover(enterHandler, leaveHandler)

hover(handler)

为匹配元素的`mouseenter`和`mouseleave`事件建立处理器。这些处理器只有在元素进入或者离开的时候才会触发，进入或者离开元素，交互不会触发事件。

参数

`enterHandler(Function)` 作为`mouseenter`处理器的函数。

`leaveHandler(Function)` 作为`mouseleave`处理器的函数。

`handler(Function)` 对于`mouseenter`和`mouseleave`事件都会调用的单个处理器。

返回

jQuery集合。

使用下面的脚本代码为页面里的第二个区域(“Outer2”和“Inner2”子区域)建立鼠标事件处理器：

```
$('#outer2').hover(report);
$('#inner2').hover(report);
```

和第一个区域一样，`report()`函数作为“Outer2”和“Inner2”的`mouseenter`和`mouseleave`的事件处理器。但是与第一个区域不同，当把鼠标指针移动到“Outer2”和“Inner2”边界之间时，这两个处理器都不会被触发(为“Outer2”)。这对于从父元素移动鼠标指针到子元素而不需要父元素处理器的情况非常有用。

现在看看在jQuery里如何创建自定义事件。

6.2.6 创建自定义事件

在jQuery里创建自定义事件非常简单，而且需要使用之前讨论的方法。自定义事件是根据可

能发生的情况和条件执行特定代码的快捷方式。假设有一条语句执行一些逻辑操作，那么可以把它们包装到一个事件处理器中，然后在需要的时候触发事件。坦率来说，事件不需要任意形式的创建，但是需要侦听和触发。这意味着，可以使用`on()`方法为自定义事件添加处理器，传递新事件的名字作为第一个参数。然后可以使用`trigger()`方法和事件名来触发事件。

一个创建和使用自定义事件的基本例子如下：

```
$('#btn').on('customEvent', function() {  
    alert('customEvent');  
});  
$('#anotherBtn').click(function() {  
    $('#btn').trigger('customEvent');  
});
```

168

在这段代码中，我们把名为`customEvent`的自定义事件添加到ID为`btn`的元素上。然后把一个事件处理器添加到ID为`anotherBtn`的元素，点击一次，触发`customEvent`事件。因为在ID为`btn`的元素上触发了`customEvent`事件，所以提示框被显示出来。

记住，jQuery不会在运行时创建自定义事件的快捷方式，所以编写如下代码：

```
$('#btn').customEvent();
```

将会抛出错误。

除了自定义事件，jQuery允许为事件添加命名空间。虽然前面已做过介绍，但是没有深入讲解，是时候详细学习了。

6.2.7 为事件添加命名空间

jQuery事件处理器提供的另外一个好处就是可以为事件处理器添加命名空间。与常见的命名空间不同(前缀命名空间)，事件是后缀命名空间，用圆点分割。如果喜欢，可以使用多个后缀来分割多个命名空间，正如`on()`方法描述的意义。通过事件分组，可以轻易实现用单元来操作事件。

例如，一个页面包含两种模式：显示模式与编辑模式。编辑模式下，事件侦听器被放置在页面的多个元素上，但是这些事件侦听器对于显示模式不太合适，需要在页面转换状态的时候删除掉。可以通过为事件添加命名空间来实现对编辑模式的操作，代码如下：

```
$('.my-class').on('click.editMode', myFunction);
```

通过把所有的事件分组到`editMode`命名空间下，后面可以作为一个整体来操作这些事件。例如，可以删除页面上所有元素的所有`editMode`命名空间下的事件：

```
$('*').off('.editMode');
```

正如我们说的，jQuery也允许我们为单个事件使用多个命名空间。在下面的例子里可以看到

这个特性:

```
$('.elements').on('click.myApp.myName', myFunction);
```

命名空间区分大小写, 所以, 如果之前的代码这样编写:

```
$('.elements').trigger('click.myapp');
```

169 ▸ 处理器就不会执行(注意小写的a)。

另外一个要强调的概念就是, 假设有下面的代码:

```
$('.elements').on('click.myApp.myName', myFunction);  
$('.other-elements').on('click.myApp', myOtherFunction);
```

假设想要执行命名空间myApp里的附加给click事件的所有处理器, 这意味着要执行myFunction()和myOtherFunction()两个函数。为此, 不需要像下面一样执行两个函数:

```
$('.elements').trigger('click.myApp');  
$('.other-elements').trigger('click.myApp.myName');
```

可以选择所有的元素设置并触发click事件, 只指定myApp命名空间下的函数:

```
$('.elements, .other-elements').trigger('click.myApp');
```

如果对比可以帮助你学习, 那么请思考多个命名空间执行或操作的例子。如果触发了命名空间下的事件, 所有附加给该事件的指定命名空间的处理器都会被执行。

掌握了这些事件处理工具之后, 就可以在第7章使用所学的知识, 以及使用这些工具的例子页面, 还有前面章节学习的其他的jQuery技术。

6.3 总结

Summary

基于目前所学的jQuery知识, 本章介绍了事件处理的本质。

我们知道, 在页面开发中处理事件是十分有挑战性的工作, 但是这对于交互性Web页面来说至关重要。这些挑战背后的事实就是在新的浏览器中有三种不同的事件模型。

有传统的基本事件模型, 也有非正式的DOM级别0事件模型。可以用某些浏览器独立操作来声明事件侦听器, 但是, 实现侦听器的函数需要不同的判断代码来处理不同浏览器之间的Event对象差异。虽然简单, 但这个模型限制了只能在某个特定的DOM元素上设置一个侦听器。

可以使用DOM级别2事件模型来避免这些不足, 这是一种更高级的和标准化的事件模型, API绑定处理器到事件类型和DOM元素上。这个事件模型的特点就是, 它只支持标准兼容的浏览器, 如Chrome、Firefox、IE9以上、Safari。

对于IE8以及之前的浏览器，基于API专用的事件模型提供了DOM级别2事件模型功能的子集，可以使用。

在一系列if语句中编写事件处理器代码——一个语句处理新浏览器，一个处理其他旧的IE浏览器——是一个早期解决问题的方法。幸运的是，jQuery提供了方法来帮助我们解决这些问题。

jQuery库提供了通用的on()方法来为元素建立任意类型的事件侦听器，也包括专门为事件定制的方法，如change()和click()。这些方法独立于浏览器，且规范了传递给处理器的常用的Event实例的属性和方法。

jQuery也提供了off()方法来删除事件处理器，允许用脚本控制事件的触发过程。如果还不够，jQuery还允许使用on()方法为不存在的元素添加事件处理器。

最后，讨论了如何创建自定义事件，以及如何为事件添加命名空间，并介绍了这个特性的好处与使用的场景。

本章演示了一些页面中使用事件的例子，也动手操作了与一些重要概念相关的例子。第7章将会介绍如何把目前学习的概念结合起来，基于jQuery来创建一个不错的Web网站程序。

DVD光盘定位器

Demo: DVD discs locator

本章内容

- jQuery选择器。
- DOM元素遍历与操作。
- 为DOM元素附加事件处理器。
- 事件委托。
- 使用自定义事件。

即使没有学习到本书的一半内容，也已经学习了一些新主题、方法和技术。前面的章节已经介绍了jQuery选择器、DOM遍历与操作及事件处理。对于每一个主题，正文中都展示了专门的例子代码。虽然每个例子对于学习单个知识点都非常棒，但是有很大的局限性。

本章将会整体介绍之前的主题，尝试使用之前学习的知识来完整地做一个网站项目。在下面的章节里，我们会开发一个具备基本完整功能的网站来管理DVD光盘的集合。现在一起看看怎么具体开发！

172

7.1 让事件开始工作

Putting events (and more) to work

假设自己是视频发烧友，专业收藏DVD光盘，目前已经有几千张存盘，这已经成为一个非常头痛的问题。光盘的组织是一个大问题，怎样才能快速找到一张DVD光盘呢？同时DVD光盘的摆放也成了一个问题。它们占据了大量的空间，如果没有解决办法，则只有睡大街了。

假设购买了DVD盒子，每个盒子可以存放100张光盘，而且比之前的摆放占用的空间更少。这样虽然你可能不需要睡公园的椅子了，但是如何组织DVD依然是一个问题。怎么才能更快地找到想要的DVD光盘而不需要翻遍所有的DVD盒子呢？

我们不能做一些喜欢按照字母顺序排序的DVD来帮助快速定位光盘。这意味着每次购买新的DVD，都需要挪动所有几十个盒子里的光盘以保证顺序排列。想象一下，如果我们购买的是

多光盘《绝世天劫》套装呢?¹

我们有一台计算机，知道如何开发Web网站程序，而且有jQuery库！通过编写一个DVD数据库程序来解决问题，帮助跟踪有什么DVD光盘以及它们存放在哪里。例子代码可以在chapter-7/dvds.html里找到。²

注意：这个演示与之前第二版中的代码相比，变化很大。如果购买了第二版（谢谢！），也强烈推荐你不要跳过本章内容。你会发现加入了很多新的内容，而且这次过滤器真的管用（在旧的版本中，一样有相同的静态结果）！

这个项目使用了Ajax调用来执行任务。虽然还没有介绍这些，但是这个概念相对于实战，其他的概念应该影响不大。因为某些浏览器的安全限制，这个演示可能会出现错误。第3章里已经提到过这个问题，但是为了方便大家学习，这里会重复介绍。

注意：由于某些浏览器的安全限制，在运行这个演示的时候可能会出现错误。为了避免这个问题，可以在Web服务器程序下运行，比如Apache、Tomcat或者IIS³，也可以为浏览器寻找一个专门的解决办法。例如，对于WebKit内核的浏览器，可以通过在命令行界面运行--allow-file-access-from-files即可。这个非常重要，它会创建一个新的进程，而不是一个新的tab选项卡，网页会在新的窗口中打开。

现在开始干活！

173

7.1.1 过滤大数据集

DVD数据库和其他程序面临一样的问题，Web交付或者其他方式。如何允许用户快速找到自己需要的信息？

可以降低技术难度进行开发，只显示一个排序的实体列表，但是，如果条目很多，那么下拉网页也会非常痛苦。此外，你想学习真正的网站开发过程？没有捷径可走！

显然，设计一个复杂的网站程序远远超出了本章的内容。因此，这里将专注于开发一个可以控制的面板，允许用户搜索过滤想要的条目。

当然，希望以后可以过滤DVD的电影的名字。但是，也希望能够根据电影发布的年代进行过滤，或者根据DVD存储的盒子，或者根据是否观看过这个电影进行查找。（这可以帮助我们回答最常见的问题——“今天晚上应该看什么？”）

¹ 影片《Armageddon》，中文译名为《绝世天劫》，是试金石电影公司于1998年出品的一部科幻灾难电影，由迈克尔·贝执导，布鲁斯·威利斯、本·阿弗莱克和丽芙·泰勒等联袂出演。——译者注

² 严格来说，这是DVD光盘管理系统，采用了Web网站的形式，包括前端页面、后台代码和数据库三个主要部分。——译者注

³ 建议使用IIS，最快捷的方式是使用Visual Studio打开例子页面，然后右键选择——在IE或者其他浏览器里打开，默认启动IIS Express简化版来运行网页。——译者注

你的第一反应也许是，这有什么大不了的！毕竟，可以使用多个字段存储来搞定，不是吗？好吧，其实没有这么快。

对于单个字段，比如像标题还好，例如，想搜索所有标题包含creature单词的电影。但是，如果只想查找1987年和1999年发布的电影呢？

为了提供强大的接口来指定过滤器，需要为DVD的不同属性指定多个过滤器。但是，在这个项目里，不允许多次指定相同的过滤器。此外，会根据需要来创建，而不是猜测需要多少个过滤器。

每个过滤器都会通过一个下拉框来实现（单选select元素）。根据字段类型的不同（string、date、number，甚至boolean），同一行的控件用来显示选择的结果。用户允许添加任意多个选择器，但是同样的过滤器只有一个，或者删除指定的过滤器。

一图胜千言，研究图7.1(a)~7.1(c)的显示过程。它们显示了初始创建的状态（图7.1(a)），添加了一个过滤器后的样子（图7.1(b)），以及添加了多个过滤器后的样子（图7.1(c)）。

174

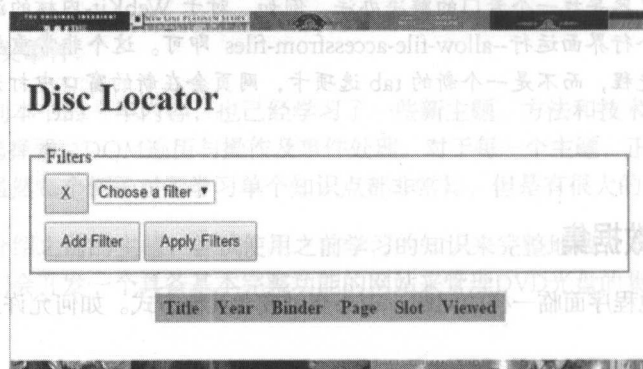


图 7.1(a) 初始状态显示的是单个没有配置的过滤器

选择一个字段后，添加限制条件

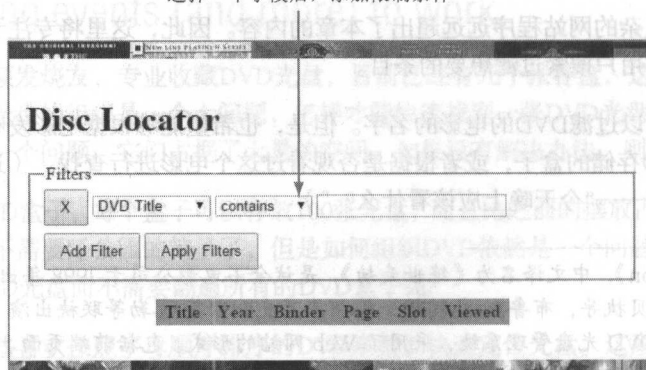


图 7.1(b) 在选择一个过滤器类型后，添加限定符控件

用户可以添加多个过滤器

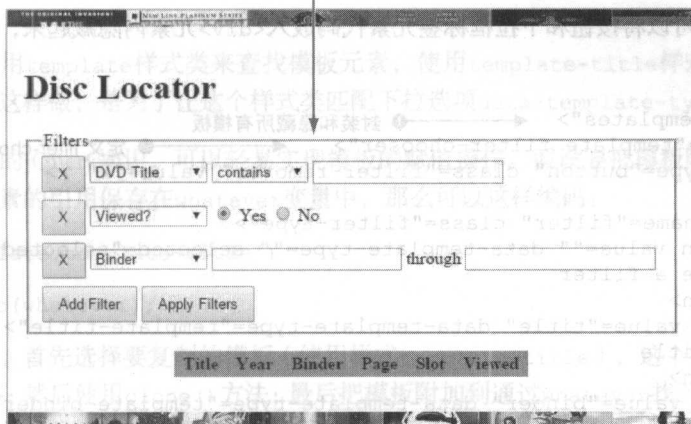


图 7.1(c) 用户可以添加多个过滤器

175

正如我们通过截图7.1(a)~7.1(c)的交互过程看到的，创建了许多新的元素。下面花点时间来看看具体的实现过程。

7.1.2 通过模板复制创建元素

可以很容易看到，要实现这种过滤控制面板，需要创建很多元素来满足不同的需求。例如，需要创建一个新的过滤器入口，当用户点击“添加过滤器(Add Filter)”按钮时，新的过滤器控件就会被选中。

没有问题！正如我们所学的，jQuery允许使用`$()`函数来动态创建新元素。虽然这个例子也要编写类似的代码，但是还需要探索一些更高级别的代码。

当动态创建元素时，所有创建这些元素的必备逻辑代码糅合到一起可能变得笨重，难以维护，即使使用了jQuery(若没有jQuery的话，那更是噩梦)。如果可以，创建一个包含复杂代码的HTML模板，然后在需要的时候复制这个模板就可。

我们望穿秋水！哈哈，jQuery `clone()`方法就是要查找的方法，它提供了这种功能。众里寻他千百度！

我们将要采用的这个方法就是创建模板标签的集合，以表示要复制的HTML代码块，在需要的时候使用`clone()`方法来创建模板的新的实例。我们不希望这些模板代码被用户看到，所以将其放到div元素里使用CSS样式隐藏起来。

作为例子，思考X按钮和下拉框组合的过滤字段。当用户点击“添加过滤器(Add Filter)”按钮时，每次都要创建这种组合的控件。jQuery代码创建这个按钮及下拉框select元素，与它的选项元素一致，代码虽有点长，但是还不至于臃肿到难以编写和维护。可以预见的是，

如果情况复杂，那么问题会迅速变得难以控制。

使用模板方法，可以将按钮和下拉框标签元素代码放入<div>元素内隐藏起来，创建标签的代码如下：

```
<div class="templates"> ① 封装和隐藏所有模板
  <div class="template filter-chooser"> ② 定义 filter-chooser 模板
    <input type="button" class="filter-remover" value="X" />

    <select name="filter" class="filter-type">
      <option value="" data-template-type="" selected="selected">
        choose a filter
      </option>
      <option value="title" data-template-type="template-title">
        DVD Title
      </option>
      <option value="binder" data-template-type="template-binder">
        Binder
      </option>
      <option value="year" data-template-type="template-year">
        Release Date
      </option>
      <option value="viewed" data-template-type="template-viewed">
        Viewed?
      </option>
    </select>
  </div>
  <!-- 更多模板在这里 -->
</div>
```

外部使用了 templates 样式的 <div> 元素作为包含模板代码的容器，设置的 CSS 样式属性为 display:none；可以阻止显示到页面上^①。在这个容器内，定义另外一个 <div>，使用样式 template 和 filter-chooser^②。通常，会使用 template 样式来区分(单个)模板，使用 filter-chooser 样式来区分特定的目标类型。很快就会看到作为 JavaScript 的钩子如何使用这些样式。

还要注意的，每个 <select> 元素里的 <option> 选项已经自定义了属性(attribute): data-template-type。我们会使用这个值来确定为选择的过滤器字段使用何种类型的过滤器控件。

根据选择的过滤器类型，会使用合适的控件来填充过滤器入口同一行的剩余空间。例如，如果模板的类型是 template-title，你想显示的是文本框控件，那么用户可以输入标题来搜索，下拉框给出了搜索字段的使用方式（包含、等于或其他等）。

为这些控件元素建立的最终的模板代码如下：

```
<div class="template template-title">
  <select name="title-condition">
    <option value="contains">contains</option>
    <option value="starts-with">starts with</option>
    <option value="ends-with">ends with</option>
    <option value="equal">is exactly</option>
```



```

</select>
<input type="text" name="title" />
</div>

```

再次，可以使用`template`样式类来查找模板元素，使用`template-title`样式标记特定的模板。我们特意这样做，是为了让这个样式类匹配下拉选项`data-template-type`的值。

使用已经学习的jQuery知识，可以轻易实现模板的赋值操作。假设要把模板附加到一个元素尾部，这个元素的引用保存在`whatever`变量中，那么可以这样编码：

```

$('div.template.template-title')
    .clone()
    .appendTo(whatever);

```

在这段代码里，首先选择要复制的模板（使用样式`template-title`），这个样式会提前设置给模板div标签；然后使用`clone()`方法；最后把模板附加到通过`whatever`找到的元素的尾部。看到了吧，为什么我们一直强调jQuery链式调用的强大优势。

177

查看下拉框`filter-chooser`元素内部的选项，会看到不同的选项值：`template-binder`、`template-year`及`template-viewed`。还要为这些过滤器类型定义控件模板，代码如下：

```

<div class="template template-binder">
    <input type="text" name="binder-min" class="numeric" />
    <span>through</span>
    <input type="text" name="binder-max" class="numeric" />
</div>

<div class="template template-year">
    <input type="text" name="year-min" class="date" />
    <span>through</span>
    <input type="text" name="year-max" class="date" />
</div>

<div class="template template-viewed">
    <label><input type="radio" name="viewed" value="true" checked="checked" />
    Yes</label>
    <label><input type="radio" name="viewed" value="false" /> No</label>
</div>

```

既然已经定义了复制策略，现在来看看主要的标签代码。

7.1.3 设置布局标签

如果再回头看看图7.1(a)，这个最初显示的DVD搜索页面非常简单：一个标题头、一个过滤器、一些按钮和显示结果的表格。下面看这个页面的实现代码：

```

<h1>Disc Locator</h1>

<form id="form-filters" action="#">

```

```

<fieldset>
  <legend>Filters</legend>
  <div id="filters"> ← ❶ 过滤器实例的容器
</div>
  <div class="buttons-wrapper">
    <input type="button" id="filter-add" value="Add Filter" />
    <input type="submit" id="filter-apply" value="Apply Filters"/>
  </div>
</fieldset>
</form>

<div id="panel-results">
  <table id="results"> ← ❷ 查询结果容器
    <tr>
      <th>Title</th>
      <th>Year</th>
      <th>Binder</th>
      <th>Page</th>
      <th>Slot</th>
      <th>Viewed</th>
    </tr>
  </table>
</div>

```

这个标签代码里没有什么特殊的地方——有吗？例如最初的下拉框代码？我们创建了一个容器来放置过滤器❶，但是其最初的状态为空，为什么？

那是因为要动态地添加新的过滤器到这个元素中——接下来就会介绍——所以为什么要在两个地方实现呢？正如你看到的，会使用代码动态地创建第一个过滤器，所以不需要提前使用静态HTML代码。另外一个要指出的地方就是，已经设置了一个表格来显示最终的结果❷。

现在已经有了简单的布局代码，而且有一些隐藏的模板可以用来快速复制生成新的元素。现在应该开始编写这些功能行为的代码了！

7.1.4 添加新的过滤器

一旦点击“添加过滤器（Add Filter）”按钮，就要为<div>添加一个新的过滤器元素，ID为filters。如果还记得如何使用jQuery建立事件处理器，那么为“添加过滤器（Add Filter）”按钮添加事件处理器的工作应该很容易实现。但是还有一些问题需要注意！

已经介绍了当用户添加过滤器时如何复制控件，而且已经有了复制控件的绝佳策略。但是，最后还是要把这些过滤条件回传给服务器，以便在数据库里过滤我们想要的结果。后台开发已经超出了本章的范畴，但是这并非意味着网站程序无法工作。我们使用一个movies.json数据文件来模拟数据库，它包含JSON数组。这个文件已经存放到本书的源码中，可以在chapter-7文件夹中找到。JSON数组里的每个对象都包含以下属性：title、year、binder、page、slot及viewed。

为了避免网站每次都加载JSON对象，可以只加载一次数据，然后存储到一个全局变量movies

里。为此,需要使用jQuery的工具函数getJSON()。到目前为止还没有介绍该函数(会在第10.3.2节里介绍),但是它做的工作就是访问包含JSON对象(数组或者任意有效的JSON格式都允许的(URL或者文件)资源,并在检索到数据时就执行一次处理器。此方法接受JSON数据参数,然后转换为JavaScript对象,传递给处理器。在处理器内部什么都不做,只是把数据保存在movies变量里,然后激发自定义事件moviesLoaded,告诉网站程序可以开始工作了。

“全局变量,我认为它是噩梦”,听你这么说过。全局变量在使用不当的时候确实是个问题。在这个例子里,这个全局变量确实表示页面级别的概念,而且它从来不会引起任何冲突,因为页面的任意代码都以同样的方式来访问单个值。然而,理想情况是,程序应该只有一个代表应用程序关注点的全局变量,这与jQuery类似,在jQuery属性里可以找到所有的方法、工具函数和属性。也可以创建一个全局变量,而不是复制给全局变量movies,例如dvdApp,用于保存电影数据和程序的方法。

179

在这个演示里,我们忽略了使用全局变量的风险,尽量简化了所有的开发工作。最终的实现代码如下:

```
var movies;
$.getJSON('movies.json', function(data) {
    movies = data;
    $(document).trigger('moviesLoaded');
});
$(document).on('moviesLoaded', function() {
    //业务逻辑在这里
});
```

在自定义事件处理器moviesLoaded内部,我们准备为“添加过滤器(Add Filter)”按钮建立另外一个点击事件的处理器。在编写必要的代码之前,需要在开头部分先编写如下代码:

```
var $filters = $('#filters');
var templatesAvailable = $('#.template', '.templates')
    .not('.filter-chooser')
    .length;
```

第一行代码是因为需要多次使用ID为filters的元素。第二行代码是因为要确认不是页面中的所有模板都在使用。一旦完成这个工作,就可以编写body的回调函数代码了:

```
$('#filter-add')
    .click(function() {
        if ($filters.find('.template:last .filter-type').val() === '') {
            return;
        }
        var filterInUse = $filters
            .children()
            .map(function() {
                return $(this)
                    .children('.template')
            })
            .length;
```

建立点击处理器

在选择过滤器之前
检查是否按按钮

查找使用的
过滤器

```

        .attr('class')
        .match(/\b(template-.+?)\b/g)[0];
    })
    .get();
    if (filterInUse.length === templatesAvailable) {
        return;
    }
    // 创建过滤器入口
    var $filterChooser = $('div.template.filter-chooser')
        .clone()
        .removeClass('filter-chooser')
        .addClass('filter');
    // 删除使用的过滤器
    $filterChooser
        .find('option[data-template-type]')
        .filter(function() {
            return filterInUse
                .indexOf($(this)
                    .data('template-type')) >= 0;
        })
        .remove();
    // 追加过滤器下拉模板
    $filterChooser.appendTo($filters);
    // 触发点击事件
    .click();

```

虽然这些代码比较复杂，但是没有用太多代码就实现了复杂的功能。下面一步一步进行讲解。

代码要做的第一件事情就是，使用jQuery的click()方法为“添加过滤器（Add Filter）”按钮①建立事件处理器。在传递给方法的函数内部，当点击按钮的时候触发事件，就会发生有趣的事情。

在执行任何操作之前，需要检查“添加过滤器（Add Filter）”按钮在用户选择过滤器之前被按下，如果错误，则需要提前终止函数②。如果因为点击了“添加过滤器（Add Filter）”按钮，接下来就要添加新的过滤器，需要创建一个新的容器来存放这个过滤器。正如之前说的，不允许出现相同类型的过滤器，所以需要在查找所有类型的过滤器③后执行它们。要查找使用的过滤器，可以使用每个元素的样式名(template-title、templateyear等)。前面章节已经介绍了两个专门的方法：map()和get()。当完成时，可以测试所有可用的过滤器是否在被使用状态：如果在使用，则终止处理器④；如果没有，则继续执行过程。

使用jQuery的clone()方法可以克隆之前设置的包含下拉框过滤器的模板。然后给它一个样式过滤器，不仅是为了CSS样式，而且是希望后面的代码中可以定位这个元素⑤。在元素创建完成后，可以使用filter()方法来排除已经被使用过的过滤器⑥。然后把克隆模板附加到主过滤器容器中，该容器在创建的时候设置ID为filters⑦。

前面的操作是处理器中最后定义的一个操作。在附加了过滤器之后，使用click()方法可触发同一个元素的点击事件⑧。在第6.2.5节“快捷方式”一节里讨论了这个版本的快捷方式。这是一个众所周知的和经常使用的执行事件处理器的方法。

为什么这么做？还记得初始化的过滤器下拉框的标签代码吗？是的，你猜对了！在页面加载时执行的处理器，第一个select元素附加到控制面板里，允许我们执行第一个选择操作。

在浏览器里加载页面，然后测试“添加过滤器（Add Filter）”按钮。注意，每次点击“添加过滤器（Add Filter）”按钮，一个新的过滤器被添加到页面。如果使用JavaScript调试器监测DOM元素（Firefox浏览器里的Firebug和Chrome浏览器里的开发工具都非常不错），就可以看到模板如何复制到容器里。

在函数（事件处理器）中使用了目前为止学习过的许多知识。这个例子又一次证明，jQuery只需要使用几行代码就可以执行复杂的操作。

但是工作还没有结束。下拉框还没有指定要过滤的字段。当用户进行选择时，需要填写过滤器容器，并为该过滤器类型添加适当的控件。

7.1.5 添加控制面板

无论用户什么时候选择下拉框中的过滤器，都要为该过滤器添加对应的控件。通过创建标签模板，可以很方便地实现这个需求，只需要选择正确的控件即可。当下拉框的值改变时，还要做一些处理工作。

现在看看你会为下拉框添加改变事件处理器做多少工作。

记住下面的代码，像第7.1.4节中的代码一样，写在自定义事件moviesLoaded的处理器内部：

```
$('#filters').on('change', '.filter-type', function() {
    var $this = $(this);
    var $filter = $this.closest('.filter');
    var filterType = $this.find(':selected').data('template-type');

    $('.qualifier', $filter).remove();
    $('#div.template.' + filterType)
        .clone()
        .addClass('qualifier')
        .appendTo($filter);
    $this.find('option[value=""]').remove();
})
```

建立改变事件处理器

复制模板

删除“Choose a filter”选项

删除任意ID的控制

使用jQuery的on()方法的优势来建立处理器，它会在正确的时候添加而不需要其他的代码。这里建立了一个改变事件处理器，使用了事件委托机制，为任意的过滤器下拉框①。这是必要的，因为在附加处理器的时候，ID为filters的div内部的过滤器还不存在。

当激发change事件处理器时，缓存包含当前元素(this)的jQuery对象，后面会多次使用。然后，收集一些信息：父过滤器容器和自定义属性data-template-type里存储过滤器类型的值。

当获取这些值时，需要删除已经在容器中存在的过滤器控件❷。毕竟，用户可以多次修改下拉框selected的值，而且不想一直不停地添加控件！当创建元素的时候，要添加样式qualifier给这些元素，以方便选择和删除这些元素。

一旦网页准备好了，就可以为正确的筛选器使用从data-template-type属性里获取的值来复制正确的模板❸。为了便于选择创建的元素，会为每个元素添加样式名qualifier（正如前面看到的代码行），而且这个元素会追加到父过滤器容器的尾部。

最后从下拉框过滤器里删除“Choose a filter”选项，因为当用户选择了一个字段值以后，再选择这个值就没有意义了。当用户选择这个选项的时候可以忽略这次的change事件，但是最好的办法就是阻止用户做没有意义的事情！

在浏览器里再次加载例子页面。尝试添加多个过滤器，然后修改选项。注意限定符如何匹配选择的字段。

现在看看删除按钮的功能……

7.1.6 删除不需要的过滤器和其他任务

我们给了用户可以修改要使用的过滤字段，但是也给了用户一个删除按钮(标签X)，可以用来完全删除过滤器。

到目前为止，我们应该知道，使用现有的知识实现这个功能应该非常简单。当用户点击按钮的时候，所有要做的就是找到最新过滤器的父容器，然后删除它！注意，在源代码里，事件处理器是使用之前的代码段通过全链式调用进行附加的。但是，为了明确概念，这里再重复一遍：

```
$('#filters').on('click', '.filter-remover', function() {  
    $(this).closest('.filter').remove();  
});
```

这里再次使用了事件委托机制，因为页面已经加载完毕，控制面板里还没有样式为filter-remover的元素。

既然所有的过滤器都设置了事件处理器，现在只剩下一件事情了：使用过滤器，然后显示结果。

7.1.7 显示结果

第7.1.6节说过，编写后台代码超出了本书的范畴，但是，我们不想给大家留个烂摊子——没有办法正常运行的程序。正如前面指出的，我们会使用JSON数组来模拟数据，这个JSON数据保存在movies.json文件里。本节将会看到表单submit的事件处理器的逻辑代码。现在看看

为 submit 事件
添加侦听器

阻止默认
的行为

```

$('#form-filters').submit(function(event) {
    event.preventDefault();

    var titleCondition = $filters.find('select[name="title-condition"]').val();
    var title = $filters.find('input[name="title"]').val();
    var binderMin = parseInt($filters.find('input[name="binder-min"]').val(), 10);
    var binderMax = parseInt($filters.find('input[name="binder-max"]').val(), 10);
    var yearMin = parseInt($filters.find('input[name="year-min"]').val(), 10);
    var yearMax = parseInt($filters.find('input[name="year-max"]').val(), 10);
    var viewed = $filters.find('input[name="viewed"]:checked').val();

    $('#tr:has(td)', '#results').remove();
    results = $.grep(movies, function(element, index) {
        (
            titleCondition === undefined &&
            title === undefined
        ) ||
        (
            titleCondition === 'contains' &&
            element.title.indexOf(title) >= 0
        ) ||
        (
            titleCondition === 'stars-with' &&
            element.title.indexOf(title) === 0
        ) ||
        (
            titleCondition === 'ends-with' &&
            element.title.indexOf(title) === element.title.length -
                title.length
        ) ||
        (
            titleCondition === 'equals' &&
            element.title === title
        )
    ) &&
    (isNaN(binderMin) || element.binder >= binderMin) &&
    (isNaN(binderMax) || element.binder <= binderMax) &&
    (isNaN(yearMin) || element.year >= yearMin) &&
    (isNaN(yearMax) || element.year <= yearMax) &&
    (viewed === undefined || element.viewed === (viewed === 'true'))
    );
});

var row;
for(var i = 0; i < results.length; i++) {
    row = '<td>' + results[i].title + '</td>';
    row += '<td>' + results[i].year + '</td>';
    row += '<td>' + results[i].binder + '</td>';
    row += '<td>' + results[i].page + '</td>';
    row += '<td>' + results[i].slot + '</td>';
}

```

查询使用的过滤器的值

根据过滤器筛选电影

清除之前的结果, 但是不包括头部

在筛选的电影上循环

格式化电影的属性
作为表格的单元

```

row += '<td>' + (results[i].viewed ? 'X' : '') + '</td>';
$('#results').append(
    $('#<tr>').html(row)
);
}
});

```

把单元格追加到行，
然后添加给 table

要给表单的submit事件添加一个函数作为事件处理器①。因为不希望浏览器在处理器代码里执行HTTP请求（没有后台代码处理），所以要阻止默认的行为②，然后执行一系列任务来获取用户选择的过滤器的值③。虽然用户不一定使用所有的过滤器，但要尝试查找所有可能的值。对于没有添加的过滤器，将会返回一个undefined(未定义)的值。

下一行代码会清理之前的结果，以保证表格干净，但是不会删除表格的头④。然后使用\$.grep()方法创建一个新的数组，只包含过滤后的电影⑤。虽然还没有介绍这个方法（会在第9.3.3节里介绍），但是它与filter()方法类似。

一旦获得查询的结果，就要考虑显示结果。为此，需要循环遍历movies数组⑥。在循环体内部格式化每个电影（movie）的属性，并转变为表格的行⑦。实际的行元素(tr)是根据实际的电影对象信息，使用jQuery()方法动态创建的。一旦创建完毕，就会附加新行到表格中⑧。

既然已经分析了处理器代码，那么继续点击“添加过滤器（Apply Filters）”按钮。开始了！集合中匹配过滤器的电影开始在页面上播放。图7.2展示了这个项目的例子结果。

加上最后一步就已经完成了这个页面，至少目前想把它用在本章作为例子代码，但是你知道的……

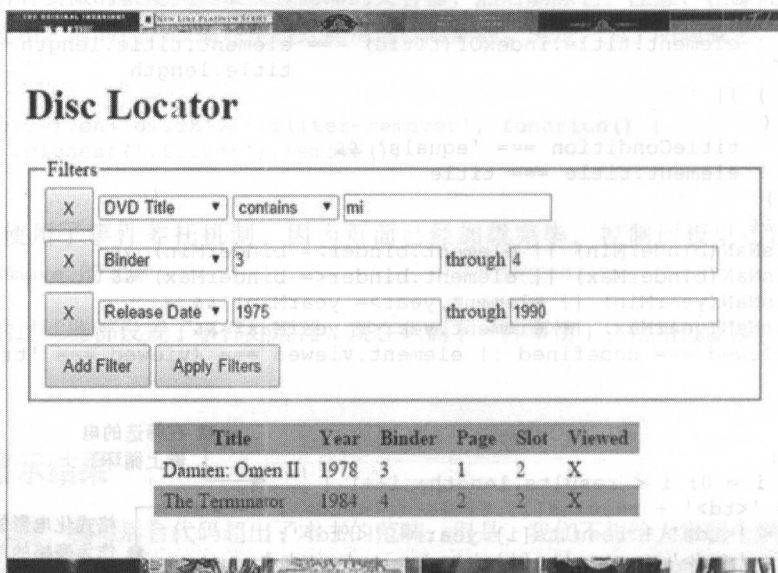


图 7.2 DVD 光盘定位器显示结果的例子

7.1.8 改进余地

从产品级的质量来看待过滤器表单，这个网站还有许多改进的空间。



下面列举了一些作为完整项目所需的额外功能，有的话会使网站更加完美。你能使用目前所学的知识来完成剩下的功能吗？

- 表单里的数据验证功能很差。例如，在表单“Submit（提交）”按钮的处理器代码里，直接把字符串转换为数字，但是更好的控制可能是有帮助的，特别是对于日期字段。

理想情况下，使用后台服务器编写前台代码，服务端处理请求——毕竟，无论如何，服务端都要做数据的验证工作。但是，这会带来差的用户体验，正如之前提到的，最好的处理错误的方法就是在发生之前阻止它。

因为解决方案监测了Event实例——有些代码没有包含在这个例子里，比如不允许用户输入其他字符，只允许用户输入数值。代码操作应该展示了本章中学习的知识，如果没有，现在是回头复习要点的好时候：

```
$('#input.numeric').on('keypress', function(event) {  
    // 字符编码 48 对应的是"0"，字符编码 57 对应的是"9"  
    if (event.which < 48 || event.which > 57) return false;  
});
```

对于支持HTML5的浏览器，有更简单的解决办法。可以强制用户使用number类型的

- 日期字段没有很好地检验。如果只允许输入有效的日期，应该怎么办？如果用户输入的开始日期比结束日期还大，怎么办？我们无法像处理数字字段一样逐个字符进行处理。
- 当添加过滤器限定符以后，用户必须点击要聚焦的字段。完全不够友好！为例子添加代码，为新生成的控件自动聚焦。
- 其中一个需求就是不允许用户有两个相同的过滤器。在当前版本里，有一种方法可以添加两个相同的过滤器，这也是示例的缺陷。你能找出并修复这个bug吗？
- 表单允许用户指定多个过滤器，但是每种类型只有一个。如何修改代码来允许用户添加多个同一类型的处理器？
- 当删除正在使用的过滤器时，过滤器的类型应该被更新。如果添加全部四种过滤器(DVD Title、Binder、Release Date和Viewed)，然后删除第一个(DVD Title)，就无法再修改现有的过滤器重新添加DVD Title了。这是因为在点击“添加过滤器（Add Filter）”按钮的时候，已经生成完毕，无法更新了。如果更新示例来处理这个问题呢？（提示：侦听包含样式filter-remover的按钮的click事件。）
- 其他可以做什么改进，比如代码的健壮性，或者接口的实用性？jQuery如何帮助我们完成任务？

186

如果你有什么新奇的想法，请访问本书在曼宁（Manning）出版社官方的主页<http://www.manning.com/derosa>，它包含了一个论坛的链接。欢迎在里面发表并讨论你的解决方法。

7.2 总结

Summary

使用目前为止学到的jQuery知识，本章开发了一个可以管理DVD光盘的Web网站。从中学到的知识就是：看起来很复杂的程序只不过是简单语句的组合。希望你能享受开发这个小网站的乐趣，它的主要目的就是强化之前学的知识概念。

187 第8章将会介绍jQuery如何实现动画和动画效果。

使用动画与特效

Energizing pages
with animations and effects

本章内容

- 不用动画实现显示和隐藏元素。
- 使用核心特效显示和隐藏元素。
- 扩展核心动画缓动函数。
- 编写自定义动画。
- 控制动画与函数队列。

在互联网发展初期,网页开发者的能力受到严重限制,不仅API少,而且脚本引擎和系统也比较弱。这种情况下使用动画和特效的想法显得很可笑,多年过去以后,唯一支持的动画方式就是使用动态GIF图片(通常都使用不当且滥用,显得页面更乱)。

今天的浏览器脚本引擎运行速度更快,要是运行在10年前的硬件设备上是无法想象的,并提供给开发者各种丰富的功能。更重要的是,新的浏览器已经使用标准化属性支持了几个CSS3模块,这些模块允许我们创建强大的动画和特效。一些属性的例子如transition、transform、filter、blur和mask。不幸的是,还存在一些问题。第一个问题就是,实际模块的实现取决于浏览器,并不是所有的浏览器都支持相同的模块。此外,浏览器支持的时间不同,跨度可能很大。第二个问题是,旧的浏览器和几个移动浏览器不支持这些模块,而且将来也不会支持。因此,如果想要创建在所有浏览器中都允许运行的动画效果,没有其他选择,只能使用JavaScript。

尽管JavaScript可以帮助我们实现这个目标,但是使用原生JavaScript并非易事。幸运的是,jQuery雪中送炭,它提供了许多简单的接口来实现动画特效。

但是,在深入学习如何向网页添加特效之前,需要思考一个问题,我们应该这样做吗?像好莱坞电影,全部是特效,没有情节,过度使用特效的页面可能会带来非常不同的和消极的反应,这并非我们期望的结果。慎重考虑用来增强页面使用性的特效,不要适得其反。此外,要记住过多的动画会导致网站性能下降,尤其是从移动设备访问网站。了解了这些注意事项,下面看看jQuery提供的动画特效。

8.1 显示和隐藏元素

Showing and hiding elements

可能最常见的动态效果就是简单地展示或者隐藏元素。虽然会介绍更美轮美奂的动画效果，但是有时候只想简单一点，弹出元素或者使之立即消失！

展示或者隐藏元素的方法要比我们期望的多得多：`show()`用于展示jQuery对象中的元素，而`hide()`用于隐藏元素。后面会讲解具体的语法，现在先看看这些方法的无参用法。

这些方法也许看起来很简单，只需要记住几点：第一，jQuery隐藏元素是通过将`style.display`修改为`none`来实现的。如果元素已经隐藏，它会继续隐藏，但是仍会返回给调用链。例如，假设有如下HTML代码：

```
<div style="display: none;">This will start hidden</div>  
<div>This will start shown</div>
```

如果编写了如下代码：

```
$('#div').hide().addClass('fun');
```

将会获得如下结果：

```
<div style="display: none;" class="fun">This will start hidden</div>  
<div style="display: none;" class="fun">This will start shown</div>
```

注意，尽管第一个元素已经隐藏，它仍然属于匹配的元素，会参与到方法调用链中。这可以从结果代码中找到证据，新的元素都设置样式为`fun`。

第二，jQuery展示对象是通过把元素的`display`属性从`none`修改为`block`或者`inline`。这个值取决于之前是否显示设置了值。如果显示设置了值，它会记住并会恢复。否则，就会基于目标元素类型的默认`display`的状态值。例如，`div`元素的`display`为`block`，而`span`的`display`为`inline`。

注意：jQuery 直到版本 1.11.0 和 2.1.0，这个机制还存在 bug。在第一个调用 `hide()` 之后，jQuery 存储了 `display` 属性的旧值在内部的一个变量里。然后，当调用 `show()` 方法后，jQuery 恢复这个值。因此，如果第一次调用 `hide()`，则 `display` 属性会设置为其他值，当调用 `show()` 时，jQuery 会恢复存储的旧值，而不是改变它。可以在这里了解详细的问题细节：<http://bugs.jquery.com/ticket/14750>。

现在已经知道了这些方法的行为，让我们看看如何与它们合作。

8.1.1 实现可收缩模块

我们对于网站配置模块（有时候作为磁贴（tile））或控制面板页面里展示各种不同信息的技术非常熟悉了。这种网站允许我们配置如何显示页面，包括移动模块、全尺寸展开模块，甚

至完全删除模块。很多模块还提供了另外一个不错的功能：允许把模块收起来放到标题栏部位，这样就可以节约页面空间，而不需要从页面完全删除它们。这看起来是一个完美的例子，可以使用前面学习的知识实现该功能。因此，接下来创建一个控制面板的模块，并且允许用户把它收缩到标题栏部位。

在深入研究编写代码之前，先来看看模块在正常状态下和卷起状态下是什么样子的。这些状态如图8.1(a)和图8.1(b)所示。

图8.1(a)展示了一个包含两个部分的模块：标题栏和内容体。内容体包含模块的数据——对这个例子是随机“Lorem ipsum”文本（关于此文本，可以通过http://en.wikipedia.org/wiki/Lorem_ipsum了解更多信息）。更有意思的标题栏包含模块的标题和一个小按钮（右下角的最小化符号），点击这个符号可以激发卷起功能。

一旦点击按钮，模块的内容就会消失，卷起收缩到标题栏。再点击一次，就会重新展开内容，恢复初始的样子。

这个例子功能的实现代码在chapter-8/collapsible.module.take.1.html里，如列表8.1所示。假如你看到文件名“take.1”部分，则可猜出我们会重构这个例子，不错，你猜对了！

190

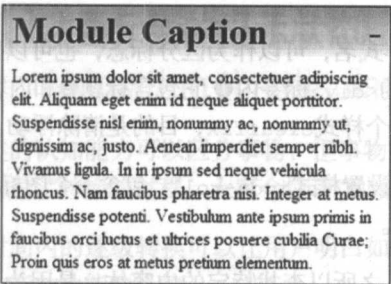


图 8.1(a) 会创建自己的控制面板模块，包含两个部分：标题栏和内容体。标题栏包含标题和最小化符号，内容体用于展示内容

图 8.1(b) 点击收缩按钮，内容体卷起收缩消失在标题栏部分

列表 8.1 收缩模块的第一个实现代码

```
<!DOCTYPE html>
<html>
  <head>
    <title>Collapsible Module - Take 1</title>
    <link rel="stylesheet" href="../css/main.css" />
    <link rel="stylesheet" href="../css/module.css" />
  </head>
  <body>
    <div class="module">
      <div class="caption clearfix">
        <h1>Module Caption</h1>
        <span class="icon-roll"></span>
      </div>
```



```


Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    Aliquam eget enim id neque aliquet porttitor. Suspendisse
    nisl enim, nonummy ac, nonummy ut, dignissim ac, justo.
    Aenean imperdiet semper nibh. Vivamus ligula. In in ipsum
    sed neque vehicula rhoncus. Nam faucibus pharetra nisi.
    Integer at metus. Suspendisse potenti. Vestibulum ante
    ipsum primis in faucibus orci luctus et ultrices posuere
    cubilia Curae; Proin quis eros at metus pretium elementum.


```

用来创建模块的标签代码非常简单。我们设置一些样式名，可以作为区分标志，也可以作为CSS样式。整个构建包含在一个设置了module样式的<div>标签内①，它包含标题②和内容③，各自的样式分别是caption和body。标题元素还有一个样式clearfix，目的是清除浮动。

为了支持收缩功能，需要把标题放置到内，并设置样式icon-roll，包含一个图标（最小化图标）。这个元素附加了一个click事件处理器④。

在click事件处理器内部，首先要定位模块的内容体。之所以查找特定的内容体，是因为页面上可能有很多模块，所以不能选择所有包含body的元素。可以找到内容体最近的模块容器，然后找到样式为body的子元素即可⑤。如果还不清楚如何查找正确的元素，那么正好可以回头复习早期章节关于查找和选择元素的内容。

一旦定位到内容体，就可以使用jQuery的is()方法来测试内容体能否显示或者隐藏⑥。如果隐藏，则使用show()方法来显示元素⑦；否则，可以使用hide()隐藏元素。这并不困难，不是吗？但正如证明的一样，可以使用更简单的方法实现！

8.1.2 切换元素的显示状态

在隐藏和显示之间切换元素的状态——正如为模块例子编写的代码一样——jQuery定义了更简单的方法叫toggle()。

使用这个方法来收缩模块，看看它如何帮助我们简化列表8.1的代码。下面列出了使用此方法

重构后的代码，粗体字是改变的代码。完整的页面代码可以在 chapter-8/collapsible.module.take.2.html 中找到。

列表 8.2 使用 toggle() 方法实现收缩

```
$( '.icon-roll' ).click(function() {  
    $(this)  
        .closest( '.module' )  
        .find( '.body' )  
        .toggle();  
});
```

192

注意，无须判断内容控件的状态来决定显示或者隐藏模块内容体，toggle() 方法会自动判断来切换状态。这个方法帮助我们大大简化了代码，而且无须存储内容的引用变量。

toggle() 方法不仅是对 show() 和 hide() 方法的简化，而且接下来你会了解该方法其他强大的用途。

让元素显示或者消失非常简单，但是有时候希望这个切换的过程不要太唐突，顺其自然就好。下面看看怎么做才可以实现平滑的过度。

8.2 动画元素的显示状态

Animating the display state of elements

人类的认知能力可以区分事物，但事物的瞬间显示和消失并不适合。如果用户眨眼了，就可能错过了转换的过程，大家会惊讶“到底发生了什么？”

短时间内的逐级转换可以让用户明白如何从一种状态转换到另外一种状态——这就是 jQuery 核心特效功能的目的。

jQuery 包含三种特效类型。

- 显示和隐藏（比 8.1 节讨论的复杂）。
- 淡入和淡出。
- 向上滑动和向下滑动。

下面逐个来看这些特效。

8.2.1 逐渐显示和隐藏元素

show()、hide() 和 toggle() 方法比之前介绍的更加灵活。当无参调用这些方法时，只是简单地操作显示 DOM 元素的状态，导致元素立即显示或者隐藏起来。但是，当传递参数时，这些效果可以延长特定的时间来完成。

现在来看看这些方法的详细语法。

hide()方法语法

hide(duration[, easing][, callback])

hide(options)

hide()

导致选中的元素隐藏起来。如果无参调用，会立即设置元素的display属性为none。如果设置了duration参数，则元素会在指定的时间通过调整高度、宽度和透明度为0，将display属性设置为none隐藏起来。

可选的(easing)函数名，指定如何转换状态。

可选回调(callback)函数，当动画完成的时候调用。在第二版中，可以通过提供包含一些选项的对象来传递给方法（后来又多了很多可用的属性）。

参数

duration(Number|String) 动画效果持续的毫秒时间长度或者使用预定义的字符串的值："slow"（等价于600毫秒）、"normal"（等价于400毫秒）或者"fast"（等价于200毫秒）。如果只指定回调函数，而忽略时间参数，则默认使用"normal"。

easing(String) 从可见状态转换到消失状态执行的easing函数名。easing函数用于指定不同点的动画步骤。如果未指定此函数，则默认值为"swing"。更多函数请参考第8.3节。

callback(Function) 当动画完成时调用的函数。无须传递参数，但是函数上下文(this)为执行动画的元素。为每个产生动画的元素执行回调函数。

options(Object) 要传递给方法的可选项，如表8.1所示。

返回

jQuery集合。

hide()方法给我们机会讨论了几点。第一点就是easing 参数，它允许指定一个easing函数。easing术语用来描述动画每一帧处理与步骤执行的方式。通过在动画过程中和当前时间位置使用精确的数学计算，就可以实现特殊的动画效果。但是有什么函数可用呢？jQuery核心库只支持两个函数：linear，它使用恒定的步骤来处理动画；swing，它处理动画开始和结束的时候速度比较慢，中间速度快。“为什么只有两个函数？”你可能会问这个问题。原因并非是jQuery想抹杀你的创造性，而是jQuery想尽可能保持自身的紧凑，把其他功能留给第三方库。我们会在第8.3节学习如何添加更多的easing(有时也称为easings)函数。

注意：使用 duration 参数的值"normal"只是一种习惯。也可以使用任何喜欢的字符串来持续 400 毫秒（除了 slow 和 fast），例如"jQuery"、"jQuery in Action"或"wow"，获得等价的效果。如果想要一探究竟，可以搜索 jQuery 库中的 jQuery.fx.speeds 属性。不过，还是推荐使用"normal"（正常模式）值，否则你的同事会疯掉。

其他值得讨论的参数是options。使用这个参数可以定义自己需要的hide()方式。表8.1列举

了允许的属性和值。

表 8.1 options 参数中允许的属性和值，按阿拉伯字母排序

属 性	值
always	当动画完成或者停止时调用的函数。传递给函数的 Promise 对象，或者是被解析的，或者是被拒绝的（第 13 章将讨论这个概念）
complete	(Function)动画完成时调用的函数
done	(Function)动画完成时调用的函数，意味着此时它的 Promise 对象被解析
duration	(String Number)动画持续的毫秒数，或者是预定义的字符串，与前面介绍的一样
easing	(String)执行从可见状态到隐藏状态的转换操作调用的函数，如前面的介绍
fail	(Function)当动画执行失败的时候调用的函数，传递的 Promise 对象被拒绝
progress	(Function)在动画的每一步之后执行的函数。无论动画的属性有多少个，函数只为每个动画元素调用一次
queue	(Boolean String)布尔值指明动画是否被放入队列中（后面会详细介绍）。如果传递的是 false，动画会立即开始，默认值为 true。如果传递的是字符串，则动画被添加到字符串指定的队列中。当使用自定义队列名称时，动画不会立即开始
specialEasing	(Object)一个或者多个 CSS 属性的地图，它的值是 easing 函数
start	(Function)动画开始时调用的函数
step	(Function)每个动画元素的每个动画属性都会执行的函数

所有这些属性都可以用来创建炫酷的特效。下面将简要介绍这个主题，开发三个不同的自定义特效。我们知道，有许多概念还很模糊，而且可能让人感到有点困惑。但是不要担心，每个概念都会详细介绍，请保持耐心。

我们已经深入学习了hide()方法及其参数，现在可以深入学习show()方法了。因为show()方法和hide()方法的参数是一样的。

show()方法语法

```
show(duration[, easing][, callback])
show(options)
show()
```

显示集合中匹配的隐藏元素。如果无参数调用，就立即把元素的display属性设置为合适的值来显示元素（block 或inline）。

如果设置了duration参数，则元素会在指定的时间段内通过调整宽度、高度和透明度来显示元素。

可选easing函数名可以用来指定状态转换的模式。

可选回调函数在动画执行完成后调用。

在第二版中，可以为方法传递一个包含多个选项的对象作为参数。

返回

jQuery集合。

正如第二个例子所示, jQuery提供了称为toggle()的快捷方法来控制元素的状态。它的语法如下, 也会忽略已经介绍过的参数。

toggle()方法语法

toggle(duration[, easing][, callback])

toggle(options)

toggle(condition)

toggle()

对于隐藏的元素执行show()方法, 对于显示的元素执行hide()方法。参见这些方法的语法描述。

在第三种形式里, toggle()方法会根据传入的条件来执行显示或者隐藏操作。如果为true, 就显示元素; 否则隐藏元素。

参数

condition(Boolean) 确定是否显示(true)或者隐藏(false)。

返回

jQuery集合。

我们来做第三个尝试, 为元素开始和结束部分也加上动画效果。学习toggle()方法后, 你可能会认为, 列表8.2中需要修改的代码就是把toggle()改成toggle('slow')即可。确实, 你猜对了。不过也没有这么快! 因为这样太简单了, 让我们再为当前的模块添加一些其他的功能。

比如, 给用户一条准确无误的线索, 想要模块标题在它的卷起状态下显示不同的图标。可以在动画完成之前进行修改, 但是, 如果在动画结束时做会更好。

196

jQuery 3:功能修改

jQuery 3中修改了hide()、show()、toggle()、fadeIn()方法, 以及本章中介绍的其他方法的行为。所有这些方法都不会重写CSS级联。这意味着一个元素如果是隐藏的——以为CSS样式里定义了display:none;——元素调用了show() (或者show()和slideDown())方法也不会展示元素。为了理解这个概念, 思考下面的元素:

```
<div class="hidden">Hello!</div>
```

现在, 假设样式表里定义了如下代码:

```
.hidden { display: none; }
```

如果使用了jQuery 3之前的版本, 则可以这样编写代码:

```
$('div').show('slow');
```

会看到一个非常不错的动画效果。

在jQuery 3中, 执行相同的代码, 将会无效, 因为没有重写CSS声明。如果想要生效, 则需要使用如下的代码替换:

```
$('#div')
    .removeClass('hidden')
    .hide()
    .show('slow');
```

或者使用下面的代码替换:

```
$('#div')
    .removeClass('hidden')
    .css('display', 'none')
    .show('slow');
```

这个修改存在争议, 因为可能导致很多网站功能崩溃。编写本书的时候, 最终版本的jQuery 3还没有发布, 这个行为可能被修改或者恢复。请查阅最新的官方文档获取最终结果。

不能在动画方法调用之后立即调用方法, 因为动画不会阻塞。动画方法后面的方法会立即执行, 甚至可能在动画开始之前已经开始了。这是使用toggle()方法的回调函数的绝佳场景。

要采取的方法就是在动画完成之后, 取代包含图标的元素的文本。如果窗体可见, 就使用减号(表示可以隐藏); 如果窗体隐藏, 就使用加号。可以通过使用CSS和content属性实现, 通过为模块添加样式名来表示收缩状态, 删除样式名来表示展开状态。因为本书并非是关于CSS的书籍, 所以跳过这个解决方案。列表8.3展示了代码要做出的必要修改。

197

列表 8.3 动画版本的模块, 修改了标题图标

```
$('.icon-roll').click(function() {
    var $icon = $(this);
    $icon
        .closest('.module')
        .find('.body')
        .toggle('slow', function() {
            $icon.text($(this).is(':hidden') ? '+' : '-');
        });
});
```

根据 body 状态, 修改图标的文本

你可以在chapter-8/collapsible.module.take.3.html中找到修改过的例子页面。

知道大家爱鼓捣东西, 所以就建立了一些可以检查操作和特效的工具方法。

8.2.2 介绍 jQuery 特效试验页面

第2章介绍的试验页面可帮助我们使用jQuery的选择器。本章也会建立jQuery特效试验页面来研究chapter-8/lab.effects.html文件里的jQuery动画特效。

在浏览器里加载这个页面, 如图8.2所示。

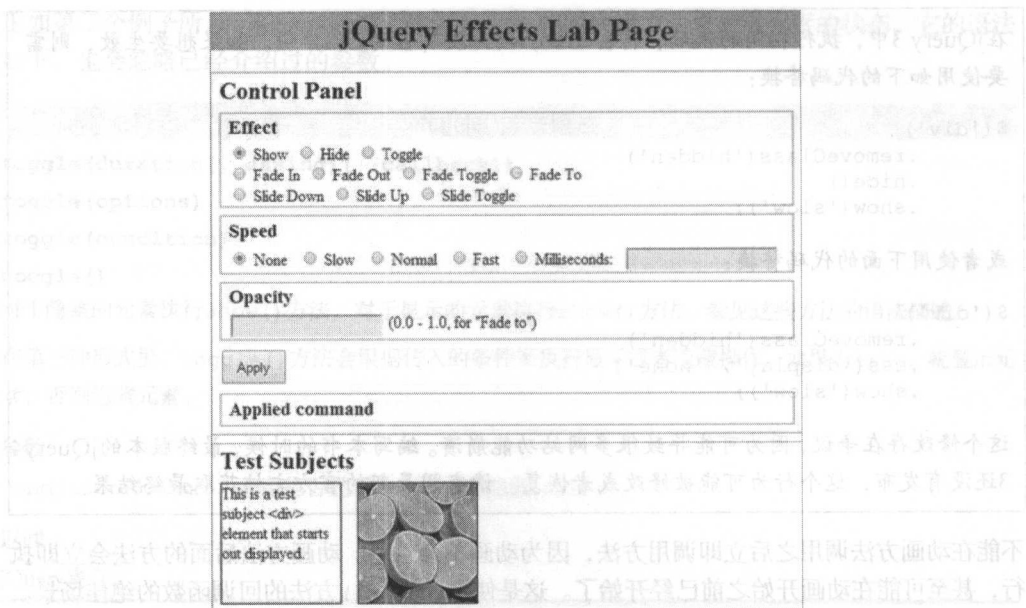


图 8.2 jQuery 特效试验页面的初始状态，它可以帮助我们监测 jQuery 特效方法的操作

这个试验页面包含两个面板：控制面板用于指定使用的特效；另外一个面板包含要测试特效的元素。



“他们是不是傻了？”你可能会这样想。“只有两个测试的目标元素。”

并不是这样，作者还没有老年痴呆。其实有四个元素，其中有两个元素最初是隐藏起来了的（一个<div>包含文本，一个是图片）。

198



我们使用这个页面演示目前为止介绍的方法。在浏览器里载入页面，按照下面的试验步骤操作。

- 试验1：在页面初始化之后，点击“应用（Apply）”按钮，会执行无参的show()方法。应用的表达式显示在“应用（Apply）”按钮的下方。注意两个最初隐藏的元素立即显示了处理。你可能好奇为什么右边的图片看起来有些模糊，那是因为透明度已经设置为50%（CSS里实际的值是0.5）。
- 试验2：选择“Hide”单选按钮，点击“Apply”按钮执行无参的hide()方法，所有测试元素都立即消失了。注意元素所在的窗口已经收缩了。这表示元素已经完全从显示中被删除而不是仅变成不可见了。

注意：当我们说元素从显示中被删除（这里回忆我们讨论的特效）时，意思是指通过设置元素的CSS的display属性为none，就不会让浏览器的布局管理器来负责。并非说元素已经从DOM树中删除了，没有动画特效会从DOM树中删除元素。

- 试验3: 选择“收缩 (Toggle)”单选按钮, 然后点击“Apply”按钮, 重复多次。你会注意到, 连续执行toggle()方法, 测试目标会收起, 然后展开, 依次往复。
- 试验4: 重新加载页面, 恢复控件到初始状态 (Firefox浏览器里可以点击地址栏, 然后按回车键。也可以点击重新加载的按钮来加载页面)。选择“收缩 (Toggle)”并点击“Apply”按钮。注意两个最初可见的元素消失了, 而且之前隐藏的两个元素出现了。这里演示了toggle()方法会针对每个独立的元素执行, 显示原来隐藏的元素, 隐藏显示的元素。
- 试验5: 在这个试验里, 我们将会进入动画领域。重新加载页面, 选择“Show”按钮, 然后选择Slow速度模式。点击“Apply”按钮, 仔细观察目标元素。两个隐藏的元素从左上角慢慢扩张显示出来, 而不是立即弹出。如果观察不够仔细, 则可以重新加载页面, 选择速度为毫秒值 (milliseconds), 设置为5000。这将延长动画的持续时间为5秒, 以留出足够观察特效的时间。
- 试验6: 选择不同的Show(显示)、Hide(隐藏)及Toggle(收缩), 以不同的速度组合, 直到你完全理解这些操作为止。

学习了jQuery特效试验页面及特效操作的知识后, 下面看看其他的动画特效。

8.2.3 逐渐消失元素

如果仔细观察show()和hide()的特效操作, 就会注意到它们都伸缩了元素的尺寸 (或者向上或者向下), 而且在增长或者搜索的时候调整了元素的透明度。下面的特效, fadeIn()和fadeOut()只影响元素的透明度。透明度为0 (完全透明) 或者1 (完全可见), 这取决于调用的方法, 它们将display的属性值设置为none或其他值 (已在第8.1中介绍了这个机制)。

除了缺少伸缩尺寸外, 这些方法与show()和hide()的工作方法类似。语法和参数也与show()和hide()的一样, 所以这里不再重复介绍。fadeIn()、fadeOut()和其他动画方法(show()、hide()、toggle())的唯一区别就是, 前者在无参数调用的时候元素不会立即产生特效。

这些内容的语法如下。

fadeIn()方法语法

fadeIn(duration[, easing][, callback])

fadeIn(options)

fadeIn()

使匹配的隐藏元素逐步修改透明度为正常值。这个透明值可能是最初设置的值, 也可能是1 (完全透明)。持续时间取决于参数duration。如果忽略duration参数, 则默认为400毫秒("normal")。只有隐藏的元素会起作用。

返回

jQuery集合。

fadeOut()方法语法

fadeOut(duration[, easing][, callback])

fadeOut(options)

fadeOut()

使匹配的显示元素逐步修改透明度为0。持续时间取决于参数duration。如果忽略duration参数,则默认为400毫秒("normal")。只有显示的元素会起作用。一旦透明度减少至0,元素就会从显示中删除。

返回

jQuery集合。

toggle()方法用来根据元素的当前状态hide()(隐藏)和show()(显示)元素,与toggle()方法的命名习惯一样,fadeIn()和fadeOut()方法也有对应的fadeToggle()。它的语法如下。

fadeToggle()方法语法

fadeToggle(duration[, easing][, callback])

fadeToggle(options)

fadeToggle()

任意非隐藏元素执行fadeOut(),任意隐藏元素执行fadeIn()。查看这两个方法的语法描述。

返回

jQuery集合。



使用jQuery特效试验页面会更有乐趣。加载试验页面,使用Fade In、Fade Out、Fade Toggle选项重新执行试验操作(不要担心Fade To,我们会详细介绍)。

要重点注意的是,当调整元素的透明度时,jQuery的hide()、show()、toggle()、fadeIn()、fadeOut()、fadeToggle()特效方法会记住元素的初始透明值。在试验中,隐藏元素之前,我们故意把图片透明度的值设置为50%。在试验各种jQuery动画特效后,元素的透明度初始值从未改变。

另外一个jQuery提供的特效方法就是fadeTo()。这个特效和之前检查逐渐隐藏特效的调整元素的透明度一样。在试验页面使用fadeTo()方法之前,先看看此方法的语法(只介绍新的参数)。

fadeTo()方法语法

fadeTo(duration, opacity[, easing][, callback])

逐级调整jQuery对象中元素的透明度,从当前值到指定opacity参数的值。

参数

opacity(Number) 元素的目标透明度，值从0到1。

返回

jQuery集合。

与其他隐藏或者显示元素时调整透明度的方法不同，fadeTo()方法不好记住元素最初的透明度。这是合理的，因为这个特效的唯一目的就是要把透明度修改为特定的值。

201



重新打开试验页面，然后显示所有的元素（应该知道怎么做），再进行下面的试验。

- 试验1：选择Fade To，然后设置速度值，再观察动画过程；4000是一个不错的值。选择设置Opacity字段（0~1）为0.1。点击“Apply”按钮，测试目标元素。
- 试验2：设置透明度为1，然后点击“Apply”按钮。所有的元素，包括最初半透明的图片，都会变成完全透明。
- 试验3：设置透明度为0，然后点击Apply元素。所有的元素都会逐渐变成不可见，但是注意，一旦它们消失，模块区域就不会收起。与fadeOut()特效不同，fadeTo()从来不会从显示中删除元素，尽管全部是不可见的。

一直试验Fade To动画特效，直到完全掌握这些行为为止。然后准备学习下一个主题。

8.2.4 向上或者向下滑动元素

另外一组显示和隐藏元素的特效——slideDown()和slideUp()——与hide()和show()特效类似，除了显示元素的时候从顶部向下滑动，隐藏元素的时候从底部向上滑动外，还有动画效果无须参数。

与之前的特效一样，滑动特效也有与收缩效果一样的方法，即slideToggle()，且方法的语法类似。

slideDown()方法语法

slideDown(duration[, easing][, callback])

slideDown(options)

slideDown()

使匹配的隐藏元素逐级增加高度显示出来。只有隐藏的元素有效。

返回

jQuery集合。

slideUp()方法语法

slideUp(duration[, easing][, callback])

slideUp(options)

slideUp()

使显示的匹配元素逐级通过减少元素高度的方式从显示中删除。

返回

202 jQuery集合。

slideToggle()方法语法

slideToggle(duration[, easing][, callback])

slideToggle(options)

slideToggle()

隐藏元素执行slideDown(), 显示元素执行slideUp()。查看各自的语法介绍。

返回

jQuery集合。



除了元素显示和隐藏的方式外, 这些特效与其他的显示和隐藏的特效一样。可以通过jQuery特效试验页面来逐步验证这些动画特效。

8.2.5 停止动画

有时想在动画开始之后停止动画效果。这可能是由用户事件触发的, 或者想执行一个新的动画效果。stop()方法可以实现这个目的。

stop()方法语法

stop([queue][, clearQueue[, goToEnd]])

停止jQuery对象中元素正在执行的动画特效。

参数

queue(String) 要停止动画所在的队列名称(会详细介绍)。

clearQueue(Boolean) 如果设置为true, 不但会停止当前的动画, 还会停止动画队列里的所有动画, 默认为false。

goToEnd(Boolean) 如果设置为true, 则完成当前动画(与停止相对), 默认为false。

返回

jQuery集合。

当使用`stop()`方法时,记住,任何动画元素已经执行的修改都会保留。此外,如果在jQuery执行比如`slideUp()`的动画时调用`stop()`,动画不会完成,元素的一部分仍然在页面上可见。如果想恢复元素的初始状态,这就是我们自己的工作,使用jQuery的`css()`或者其他类似的方法可恢复元素的初始样式。

为了避免元素执行部分动画效果,可以为`goToEnd`方法传递`true`参数。如果想删除动画队列里的所有动画,或者在当前动画完成后修改CSS的值,则可以调用`stop(true, true)`方法。

通过指定当前动画,我们指的是,如果链式调用了三个动画效果,并且在第一个动画执行的时候调用了`stop(true, true)`,那么样式就是第一个动画执行时的值,后两个动画不会执行(从动画执行队列里删除了)。

在某些情况下,当动画停止的时候,可能要修改CSS属性,比如动画完成的时候。此时可以调用`finish()`。

finish()方法语法

finish([queue])

停止jQuery对象中当前执行的动画,删除所有队列中的动画,然后立即设置CSS属性为目标值。

参数

`queue(String)` 停止动画的队列名称。如果没有指定,则默认为jQuery使用的`fx`队列。

返回

jQuery集合。

为了方便比较`stop()`和`finish()`的不同,我们创建一个示例,可以在chapter-8/stop.vs.finish.html和JS Bin(<http://jsbin.com/taseg/edit?html,js,output>)中找到。

除了这两个方法,还有一个全局方法叫`jQuery.fx.off`,可以用来完全禁用所有的动画效果。设置标志位为`true`会导致动画立即消失。另外一个处理动画的jQuery标志是`jQuery.fx.interval`。将在第9章中详细介绍如何使用这个标志。

我们已经学习了jQuery核心库提供的动画效果,现在看看如何编写自己的动画效果。

8.3 为jQuery添加更多 easing 函数

Adding more easing functions to jQuery

第8.2节学习了`easing`参数和jQuery提供的`easing`函数,即`linear`和`swing`。jQuery提供的核心特效非常少,主要是为了最小化jQuery内核副作用。但是,这并不意味着我们不能使用第三方的库来获取更多的`easings`(`easing`函数统称为`easings`)。jQuery的`Easing`插件(<https://github.com/gdsmith/jquery.easing>)和jQuery UI库(<http://jqueryui.com>)提供了额外的转

204 换和动画特效。plugin（插件）一词我们应该十分熟悉，它指的是为现有的软件、框架或者库添加特定功能的组件。我们将在第12章介绍插件和如何创建jQuery自定义扩展的插件。

注意：jQuery UI 库包含用户界面交互、特效、组件及主题的集合。它非常不错，值得花时间了解（在阅读完本书后）。¹

当有多个选择以后，人们又喜欢问哪个才是最好的。通常来说，并没有最好的解决方案，但是应该根据特定的使用情况选择。为了帮助我们选择库，假设添加特效，而已经因为其他原因项目使用了该库（比如为了使用它提供了微件）；否则，就应该使用插件。原因是两者都提供了相同的easing函数，但是jQuery Easing 插件更加轻量级（压缩版本只有3.7 KB），而且只专注于一个功能；而jQuery UI包含的并非只有一个easings。为了完整性，必须提一下jQuery UI的下载页面，它允许我们自定义下载库，但只包含我们需要的模块，这样会减小不必要的库。

jQuery Easing 插件和jQuery UI库新增了30多个新的easing函数，如表8.2所示。

表 8.2 jQuery Easing 插件和 jQuery UI 库新增的 easing 函数

easeInQuad	easeOutQuint	easeInOutCirc
easeOutQuad	easeInOutQuint	easeInElastic
easeInOutQuad	easeInExpo	easeOutElastic
easeInCubic	easeOutExpo	easeInOutElastic
easeOutCubic	easeInOutExpo	easeInBack
easeInOutCubic	easeInSine	easeOutBack
easeInQuart	easeOutSine	easeInOutBack
easeOutQuart	easeInOutSine	easeInBounce
easeInOutQuart	easeInCirc	easeOutBounce
easeInQuint	easeOutCirc	easeInOutBounce

jQuery Easing插件有一个特性值得讨论。它存储了jQuery的swing函数，是默认的easing函数，在jswing名下。此外，它重新定义了swing，行为与easeOutQuad函数一样。因为这个修改，后者变成默认的easing函数。

205 要在页面里使用jQuery Easing插件，有两种方法。第一种简单的方法就是直接使用CDN。应该记得我在第1章里讨论jQuery内容的时候介绍了一个CDN服务器。

第二种方法就是从存储库下载插件，然后保存到页面访问的文件夹中。例如，把插件文件放在名为js的文件夹中，可以通过编写如下代码使用：

```
<scriptsrc="js/jquery.easing.min.js"></script>
```

¹除了官方网站，还有一本 T.J.VanToll 编写的书《jQuery UI in Action》(Manning, 2014)。这是一本好书！—译者注

为了在页面中使用jQuery UI, 同样也有两种方法: CDN和本地文件。可以直接在页面包含jQuery CDN, 假设使用的是1.11.4版本, 可以这样编写代码:

```
<script src="http://code.jquery.com/ui/1.11.4/jquery-ui.min.js"></script>
```

如果是本地存储文件, 假设库放在名为js的文件夹中, 则可以这样编写:

```
<scriptsrc="js/jquery-ui-1.11.4.min.js"></script>
```

现在, 既然已经知道如何在页面中使用jQuery Easing 插件和jQuery UI库, 那么可以使用提供的easing函数了。使用很简单, 只需传递要使用的easing函数名字作为easing参数即可。

要快速了解表8.2中列举的easing函数的历史, 可以查看<http://api.jqueryui.com/easings/>文档。但是, 按照要求, 只简要看看这些文档还不够。因此, 我们创建了试验页面允许你使用本章中介绍的方法来观察这些动画特效。



新的试验页面如图8.3所示, 称为jQuery高级特效试验页面(jQuery Advanced Effects Lab Page), 可以在chapter-8/lab.advanced.effects.html里找到本书的源码。

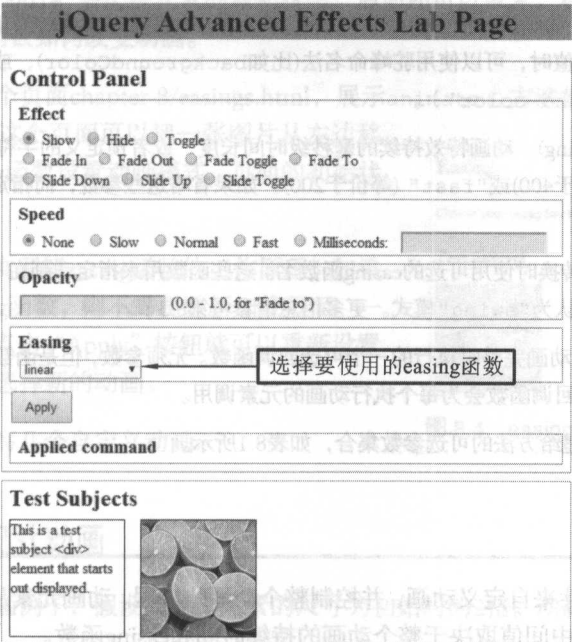


图 8.3 jQuery 高级试验页面的初始状态

玩转试验页面, 直到完全明白每个easing函数修改页面元素的动画特效区别。

目前为止, 我们已经介绍了预定义的动画效果, 但是有时想要创建自己的动画。下面看看如何实现。

8.4 创建自定义动画

Creating custom animations

第8.3节介绍的使用第三方库扩展新的easing函数很简单。创建自定义动画也如此。

jQuery暴露了`animate()`方法,该方法允许我们在一系列元素上使用自定义动画。下面看看这个方法的语法,包含参数描述。从上次介绍参数的含义到现在已经过去很久,所以,即使有相同的参数,也会重复介绍。这样就不需要再翻很多页回去查找资料了。

206

`animate()`方法语法

`animate(properties[, duration][, easing][, callback])`

`animate(properties[, options])`

通过设置元素的`properties`属性(`duration`、`easing`函数、`callback`(回调)函数)控制集合元素的动画效果。回调函数在动画执行完成的时候执行。除了`properties`,还可以使用参数集合`options`设置。

参数

`properties(Object)` 动画执行依据的CSS属性和值的对象。动画会从元素的当前值转换到对象中指定的样式值。当指定多个值时,可以使用驼峰命名法(比如`backgroundColor`),或者使用以横线分割的属性名称(`'background-color'`)。

`duration(Number|String)` 动画特效持续的毫秒级时间长度,或者预定义的字符串: `"slow"` (等价于600)、`"normal"` (等价于400)或`"fast"` (等价于200)。如果省略这些参数,而指定回调函数,则默认使用速度`"normal"`模式。

207

`easing(String)` 执行转换时使用可选的easing函数名。这些函数用来指定动画在不同点的执行步骤。如果没有指定参数,则默认为`"swing"`模式。更多信息请参考第8.3节。

`callback(Function)` 动画完整时执行的一个可选回调函数。无须参数,但是函数上下文(`this`)设置的是当前执行动画的元素。回调函数会为每个执行动画的元素调用。

`options(Object)` 传递给方法的可选参数集合,如表8.1所示。

返回

jQuery集合。

可以通过CSS样式属性来自定义动画,并控制整个动画的过程。动画元素从最初状态逐步向目标样式过渡。样式的中间值取决于整个动画的持续时间和easing函数。

指定的值可以是绝对值,也可以是相对于起点的值。要指定相对值,可使用`+=`或者`-=`符号,表示加或者减运算。

默认情况下,动画会添加到执行队列;添加多个动画会采用顺序执行模式。如果希望并行运行动画,则可以设置`queue`为`false`。

使用动画改变CSS属性，有一个限制就是接受数值类型参数，因为有一个从初始值到目标值的转换工作。数值限制是可以理解的——如何逐渐改变非数值属性值，比如background-image？对于表示尺寸的值，jQuery默认的单位是像素，但是也可以使用em或者百分比作为单位，加上em或者%后缀。

注意：CSS 里可以用动画修改元素的 color 属性，但是不能直接使用 jQuery 的 animate() 方法修改，除非使用扩展插件 jQuery.Color(<https://github.com/jquery/jquery-color>)。

经常使用的动画样式属性包括top、left、width、height和opacity。但是，如果对动画特效有帮助，也可以使用动画修改样式的其他属性，比如font-size、margin、padding和border。

此外，对于有些特殊的属性值，也可以指定字符串"hide"、"show"或"toggle"；jQuery会计算对应的数值。例如，使用opacity属性的"hide"值，会导致元素的透明度为0。使用任意这些字符串之一都可以显示或者从显示中删除元素而添加动画效果（像hide()和show()方法）。◀ 208 应该注意的是，"toggle"会记住元素的最初状态，后面还可以恢复。在继续学习之前，应该先完全理解easing函数如何改变动画。

为此，我们创建一个页面chapter-8/easings.html，展示animate()方法如何与jQuery Easing插件一起工作。使用这个页面可以把一张图片从左边移到右边，使用easing函数设置动画模式。页面的初始状态如图 8.4所示。

操作这个页面，直到理解easing函数的工作原理。如果要测试更多的easing函数，则不需要重新加载页面。一旦一个动画结束，点击“Apply”按钮就可以重新设置图片的位置，开始选择新的动画。

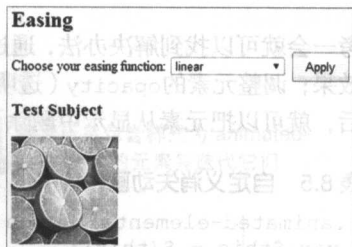


图 8.4 easings.html 页面的初始状态

现在试试自己来编写几个自定义动画。

8.4.1 自定义尺寸动画

思考一个简单的动画例子，假设要调整元素的尺寸为初始时的2倍。动画代码如列表8.4所示。

注意：这个特殊的动画在新的浏览器里可以通过专门的 CSS 来实现 transform: scale(2)。正如本章开始部分所示，新的浏览器支持许多新的标准，但是应该知道这些新特性只在新浏览器里可用。

列表 8.4 自定义尺寸动画

```
$('.animated-elements').each(function() {  
  var $this = $(this);  
  $this.animate({
```

← ① 迭代每个匹配的元素


```

width: $this.width() * 2,
height: $this.height() * 2
},
2000
);
});

```

指定每个目标值

设置持续毫秒级时间

为了实现这个动画效果，需要迭代遍历所有具备样式`animatedelements`的元素，并通过使用jQuery的`each()`方法①。这样可以为每个元素单独使用动画。这一点非常重要，因为需要根据每个元素的尺寸来设置属性值②。如果要为单个元素启用动画效果（比如使用ID选择器）或者集合中的元素全部使用同一个尺寸，就不需要使用`each()`方法，可以直接为所有的元素集使用动画。

回调函数传递给`each()`方法，`animate()`方法每次应用到一个元素上。可以使用`this`且设置宽度和高度属性值等于初始值的2倍来访问当前元素。结果就是经过2秒（设置为2000毫秒持续时间③）的动画过程，jQuery对象中的每个元素都会变成原来大小的2倍。

我们会针对本节中的例子提供一个演示页面，但是此刻再扩展学习一下。

8.4.2 自定义消失动画

假设在动画完成后想删除动画，那么用来操作元素的动画要让元素从显示中消失才行。

思考一会就可以找到解决办法，通过调整元素位置的`top`属性，可以移动元素到底部来模拟消失效果；调整元素的`opacity`（透明度）属性也可以让元素看起来消失了。最后，完成所有步骤后，就可以把元素从显示中删除了。可以使用列表8.5的代码来实现。

列表 8.5 自定义消失动画

```

$('.animated-elements').each(function() {
    var $this = $(this);
    $this
        .css('position', 'relative')
        .animate({
            opacity: 0,
            top: $(window).height() - $this.height() -
                $this.position().top
        },
        'normal',
        function() {
            $this.hide();
        }
    );
});

```

选择所有样式为 `animated-elements` 的元素

从静态流排除元素

计算跌落距离

当动画完成的时候执行

从显示里删除元素

再对第8.4.1节的例子做进一步修改。虽然可以迭代集合里的每个元素，但是这次要调整元素的位置和透明度。要调整元素，相对于初始位置的`top`值，首先要修改CSS样式的位置属性①。

然后指定目标透明度为0，以及一个`top`值。我们并不想把元素移到窗口底部下面太远。太远

会导致页面滚动条变化，而误导用户。我们也不想把用户的注意力从动画上转移——吸引用户注意力才是动画的首要目的！使用高度、元素的垂直位置，以及窗口的高度来计算元素应该下移到页面下面多远❷。当然，这个考虑只有在元素和页面底部有足够大空间的时候才有意义。

动画结束时，需要从显示中删除元素，所以先指定一个回调函数❸，然后为元素❹调用hide()方法。

注意：相比最初的需求，我们做了更多的尝试。当动画完成时，可以在回调函数里执行特定的操作。如果指定透明属性的值为"hide"而不是0，那么动画结束时删除元素的工作将会自动完成，而不需要借助回调函数。

现在来尝试更多类型的消失特效。

8.4.3 自定义烟雾动画

假设想要的特效是，元素好像一缕青烟一样消失在空气中，而不是消失在页面底部。为了模拟这种特效，可以结合使用尺寸和透明度特效，在元素退去的时候放大元素的尺寸。这个特效的问题是，元素增加的时候不会固定住左上角。元素在增加的时候，其中心位置不变。所以，除了大小，还需要调整元素的位置，以作为动画的一部分工作。列表8.6展示了烟雾特效的代码。

列表 8.6 自定义烟雾动画

```
$('.animated-elements').each(function() {
  var $this = $(this);
  var position = $this.position();
  $this
    .css({
      position: 'absolute',
      top: position.top,
      left: position.left
    })
    .animate({
      opacity: 'hide',
      width: $this.width() * 5,
      height: $this.height() * 5,
      top: position.top - ($this.height() * 5 / 2),
      left: position.left - ($this.width() * 5 / 2)
    },
    'fast'
  );
});
```

选择所有包含样式为 animated-elements 的元素并迭代它们 ❶

❷ 从静态流排除元素

❸ 调整元素大小、位置和透明度

在这个动画里，我们选择所有具备样式animated-elements的元素，并且迭代所有元素❶。然后在元素增加到原来5倍大小的时候把透明度减小到0，并调整其位置的一半，结果元素还保留在原来的位置❷。我们并不希望动画元素周围的元素被排挤出去，当动画元素变大的时

候，所有通过设置元素的位置从相对值改成绝对值^②。因为为透明度指定了hide值，所以元素会在动画完成后自动隐藏起来（从显示中删除）。三个自定义动画效果都可以在chapter-8/custom.effects.html文件里找到，如图8.5所示。

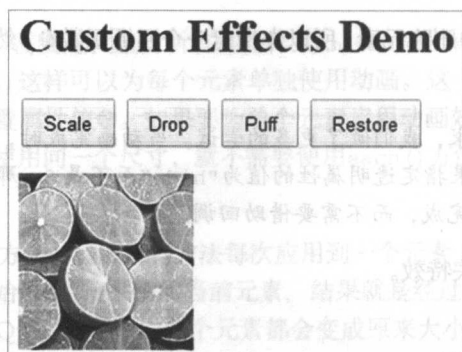


图 8.5 使用例子页面不同的按钮(scale、drop、puff)可以观察不同自定义特效的动画

虽然很想向你展示这些动画特效的行为，但是你也明白截图的局限性。图8.6展示了烟雾效果过程中的一个场景。希望读者可以自己动手试验并观察这些特效的行为过程。

到这里为止，我们学习过的所有例子使用的都是单个动画方法。下面讨论当使用多个动画时的工作原理。

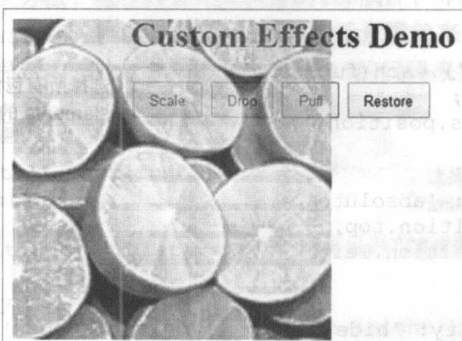


图 8.6 烟雾效果扩展和移动的图片，同时降低了图片的透明度

8.5 动画与排队

Animations and queuing

虽然已经学习了使用单个动画方法如何调用特效元素的多个属性，但是还没有学习同时调用多个动画方法时的动画行为。本节将学习动画之间如何协作运行。

8.5.1 同时动画

在执行以下代码的时候，你期望达到什么效果？

```
$('#test-subject').animate({left: '+=256'}, 'slow');  
$('#test-subject').animate({top: '+=256'}, 'slow');
```

我们知道，当页面执行`animate()`方法时不会阻塞当前线程，其他动画方法也一样。可以通过试验下面的代码来证明这一点：

```
console.log(1);  
$('#test-subject').animate({left: '+=256'}, 'slow');  
console.log(2);
```

执行这段代码，会看到控制台上打印出“1”和“2”，一个接一个，不需要等待动画执行完成。如果要证明这是真的，可以查看chapter-8/asynchronous.animate.html或者JS Bin给我们创建(<http://jsbin.com/pulik/edit?html,js,console,output>)的例子代码。

当同时调用两个动画方法时，你期望产生什么结果？因为第二个方法不会被第一个方法阻塞，因此两个动画会同步触发（或者相差几毫秒），动画元素的特效可能是两个方法的结合。这种情况下，因为一个特效是调整`left`的风格属性，另外一个特效是调整`top`的风格属性，所以可能认为元素会沿蜿蜒对角线运动。

我们来实证一下。在文件[chapter-8/revolutions.html](#)里，设置两个图片（一个启用动画）和一个按钮开始动画。此外，将使用控制台来输出结果。

图8.7展示了最初的状态。

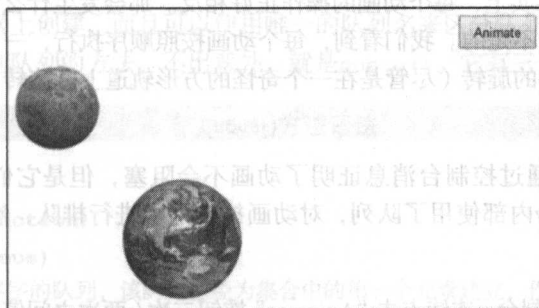


图 8.7 页面的初始状态，可以观察多动画同时执行的效果

◀ 213

“Animate”按钮调用的监控代码如列表8.7所示。

列表 8.7 监控同时执行多个动画

```
function formatDate(date) {  
    return (date.getHours() < 10 ? '0' : '') + date.getHours() +  
    ':' + (date.getMinutes() < 10 ? '0' : '') + date.getMinutes() +  
    ':' + (date.getSeconds() < 10 ? '0' : '') + date.getSeconds() +  
    '.' + (date.getMilliseconds() < 10 ?
```

```

    '00' : (date.getMilliseconds() < 100 ? '0' : '') +
    date.getMilliseconds();
}

$('#button-animate').click(function() {
    var $moonImage = $('img[alt="moon"]');
    console.log('At ' + formatDate(new Date()) + ' 1');

    $moonImage.animate({left: '+=256'}, 2500);
    console.log('At ' + formatDate(new Date()) + ' 2');

    $moonImage.animate({top: '+=256'}, 2500);
    console.log('At ' + formatDate(new Date()) + ' 3');

    $moonImage.animate({left: '-=256'}, 2500);
    console.log('At ' + formatDate(new Date()) + ' 4');

    $moonImage.animate({top: '-=256'}, 2500);
    console.log('At ' + formatDate(new Date()) + ' 5');
});

```

为定义按钮
● 点击处理器

在按钮点击处理器代码里❶，一次激发了四个动画，穿插调用日志方法`console.log()`来记录什么时候激发动画调用。

打开页面，点击“Animate”按钮。正如我们所料，控制台消息“1”到“5”立即出现，如图8.8所示，每个动画开始相差几毫秒。

```

At 03:36:19.972 1 revolutions.html:56
At 03:36:19.979 2 revolutions.html:58
At 03:36:19.980 3 revolutions.html:60
At 03:36:19.980 4 revolutions.html:62
At 03:36:19.980 5 revolutions.html:64

```

图 8.8 控制台快速连续显示消息，证明动画方法不会阻塞，直到完成为止

但是动画呢？如果仔细查看列表8.7的代码，就可以看到有两个动画修改了`top`属性，而另外两个动画修改了`left`属性。事实上，每个动画的操作正好相反。那会发生什么？会不会互相抵消？图片还是老样子不变？不是的。我们看到，每个动画按照顺序执行，一个接一个，所以月亮围着地球做了一个有序的旋转（尽管是在一个奇怪的方形轨道上做旋转，这个轨道会让开普勒先生的脑袋爆炸）。

❷ 这是怎么回事？已经通过控制台消息证明了动画不会阻塞，但是它们确实是按顺序执行的。这是因为在jQuery内部使用了队列，对动画执行请求进行排队，然后按照顺序执行每个动画。

重新刷新页面，清空控制台，连续点击“Animate”按钮三次（两次之间停顿，避免导致双击）。我们注意到，立即出现15条消息在控制台里，表示点击处理器已经执行了三次，而且月亮围着地球转了三圈。12个动画请求被jQuery排队后，按照顺序执行，因为jQuery库为每个元素维护了一个名为`fx`的队列。

更强大的是，jQuery允许自己创建自己的队列，不仅是为动画，而且可以为任何其他目的创建对象。我们来学习一下。

8.5.2 排队执行函数

排队动画串行执行是函数队列的常见用法。但是真的有好处吗？毕竟，动画方法允许完成后回调，所以为什么不在回调函数里激发下一个动画呢？

添加函数到队列

我们来复习列表8.7的代码(为了简洁，只保留`console.log()`调用代码):

```
var $moonImage = $('img[alt="moon"]');
$moonImage.animate({left: '+=256'}, 2500);
$moonImage.animate({top: '+=256'}, 2500);
$moonImage.animate({left: '-=256'}, 2500);
$moonImage.animate({top: '-=256'}, 2500);
```

对比不使用函数队列的代码，使用完成后的回调函数：

```
var $moonImage = $('img[alt="moon"]');
$moonImage.animate({left: '+=256'}, 2500, function(){
    $moonImage.animate({top: '+=256'}, 2500, function(){
        $moonImage.animate({left: '-=256'}, 2500, function(){
            $moonImage.animate({top: '-=256'}, 2500);
        });
    });
});
```

并不是说回调代码更加复杂，但是也很难说最初的代码是否更易于阅读（开始编写的）。如果回调函数的代码体非常复杂……那么好吧，可以轻易看出排队代码大大简化了代码的复杂度。

215

队列可以在任意元素上创建，而且可以使用唯一的队列名来区分队列（除了为动画特效保留的`fx`）。添加函数到队列的方法，不出意外，就是`queue()`。它有三个变量。

queue()方法语法

queue([name])

queue([name], function)

queue([name], queue)

第一种形式返回传递名字的队列，该队列已经为集合中的第一个元素建立，作为函数数组。

第二种形式为集合中所有匹配的元素添加传递的函数到命名队列的尾部。如果命名队列在元素上不存在，就会创建该队列。

最后一种形式取代匹配元素上任意存在的队列，使用传递的队列。

当忽略`name`时，默认的队列是`fx`。

参数

`name(String)` 获取、添加、取代队列的名字。如果忽略，则默认是`fx`。

queue()方法语法

`function(Function)` 添加到队列尾部的函数。当调用时，函数上下文(`this`)是建立队列的DOM元素。函数只传递一个参数`next`。`next`是另外一个函数，当调用时，下一个项目自动出队，然后保持队列移动。

`queue(Array)` 函数数组，取代命名队列里已经存在的函数。

返回

第一种形式返回函数数组，其余的形式返回jQuery集合。

`queue()`是最常用的向命名队列尾部添加函数的方法，且它可以用来从现有队列中获取函数，或者取代队列中的函数列表。注意，第三种形式虽然能传递函数数组给`queue()`，但是不能用来添加多个函数给队列，因为任意已经存在的队列函数被删除了。为了向队列里添加函数，需要使用第一种形式从队列中取出已有的函数，合并新的函数，然后把合并的函数结果集合重新使用第三种形式`queue()`添加到队列里。

“没有例子吗？”你可能会问。使用`queue()`方法，可以添加新的动画到队列尾部，但是你可能认为还没有讨论如何执行它们？让我们来看看如何设置demo。

执行队列函数

216 队列化执行函数没有用，只有实际执行它们才行。输入`dequeue()`方法。

dequeue()方法语法

`dequeue([name])`

在jQuery对象里为每个元素删除命名队列里最前面的函数，然后为每个元素执行它们。

参数

`name(String)` 要删除函数的队列名。如果忽略，则默认使用`fx`。

返回

jQuery集合。

当调用`dequeue()`时，会执行jQuery对象里每个元素的队列中最前面的函数，并使用当前元素作为函数上下文(`this`)。思考列表8.8的代码，可以在文件chapter-8/manual_dequeue.html里找到。

列表 8.8 在多个元素中队列和出队函数

```
<!DOCTYPE html>
<html>
  <head>
    <title>Manual Dequeue</title>
    <link rel="stylesheet" href="../../css/main.css" />
    <style>
      button
      {
        display: block;
```



```

        margin: auto;
    }
</style>
</head>
<body>
    <button>Dequeue</button>

    <script src="../js/jquery-1.11.1.min.js"></script>
    <script>
        var $images = $('img');

        $images
            .queue('chain', function() {
                console.log('First: ' + $(this).attr('alt'));
            })
            .queue('chain', function() {
                console.log('Second: ' + $(this).attr('alt'));
            })
            .queue('chain', function() {
                console.log('Third: ' + $(this).attr('alt'));
            })
            .queue('chain', function() {
                console.log('Fourth: ' + $(this).attr('alt'));
            });

        $('button').click(function() {
            $images.dequeue('chain');
        });
    </script>
</body>
</html>

```

建立四个队列函数 ①

每次点击 ② 出队函数

217

在这个例子中，有两个图片，并为名为chain的队列添加了函数①。在每个函数内部，在控制台上打印每个元素的alt属性，这个元素作为当前函数的上下文，以及函数在队列里的顺序。这样，就可以区分哪个函数在执行，并且从哪个元素的队列里开始。所以，此时，无须为图片做任何特殊处理（它们不会动）。点击“Dequeue”按钮，按钮的处理器②会执行单个dequeue()方法。继续点击，然后观察控制台的消息，如图8.9所示。

我们看到队列里为图片添加的第一个函数被激发了两次：一次是地球图片，一次是月亮图片。点击“Dequeue”按钮多次，会从队列中一次删除一个后续的函数，然后执行这些函数，直到队列被清空为止。

在这个例子中，出队函数是人为控制的——需要点击四次按钮（调用四次dequeue()方法）来执行所有的四个函数。我们经常希望能激发队列里的所有函数。此时，一种常用的方法就是在队列函数内部调用dequeue()或者使用传递给队列函数的next参数，以创建链式调用。

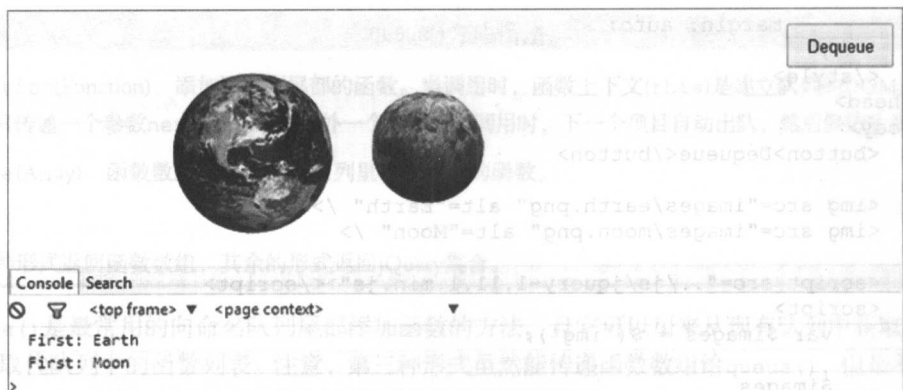


图 8.9 点击“Dequeue”按钮会导致函数的单个队列实例被激发，每个被创建事件的图片执行一次

思考下面的代码，修改列表8.8中使用的两种方法：

```
var $images = $('img');
```

```
$images
```

```
.queue('chain', function(next) {
  console.log('First: ' + $(this).attr('alt'));
  next();
})
.queue('chain', function(next) {
  console.log('Second: ' + $(this).attr('alt'));
  next();
})
.queue('chain', function() {
  console.log('Third: ' + $(this).attr('alt'));
  $(this).dequeue('chain');
})
.queue('chain', function() {
  console.log('Fourth: ' + $(this).attr('alt'));
});
```

chapter-8/manual.dequeue.html代码的修改版本包含之前的代码，可以在文件chapter-8/automatic.dequeue.html里找到。

在浏览器中打开页面，点击“Dequeue”按钮。注意单击如何触发执行整个队列函数的调用链，如图8.10所示。注意最后一个添加到队列的函数没有调用dequeue()，因为此时，队列已经清空了。

使用dequeue()，可以执行队列顶部的函数。但是有时候希望删除所有存储在队列中的函数，而不希望执行它们。

清除不执行队列函数

如果想清空所有队列函数而不执行它们，则可以使用clearQueue()方法。

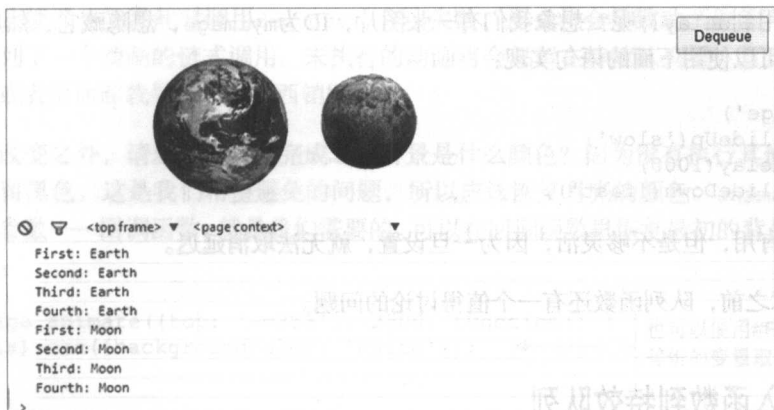


图 8.10 点击“Dequeue”按钮会引起队列里所有函数执行它们建立的元素

219

clearQueue()方法语法

clearQueue([name])

删除队列里所有未执行的函数。

参数

name(String) 删除函数队列的名字。如果忽略，则默认名为fx。

返回

jQuery集合。

与stop()动画方法类似，clearQueue()适合所有队列函数，而不仅仅是动画特效函数。

有时候，我们是想延迟执行队列函数而不是清空队列。下面讨论一下可能性。

延迟队列函数

另外一个面向队列的活动，可能要执行的就是延迟执行队列函数。delay()方法就是为此而生的。

delay()方法语法

delay(duration[, queueName])

在命名队列里为所有未执行的函数添加延迟。

参数

duration(Number|String) 延迟时间的毫秒数，或者字符串“fast”、“normal”、“slow”，分别表示200毫秒、400毫秒和600毫秒。

queueName(String) 延迟执行函数的队列。如果忽略，则默认为fx。

返回

jQuery集合。

什么时候会用到`delay()`呢？想象我们有一张图片，ID为`myimage`，想隐藏它，然后在1秒后显示出来。可以使用下面的语句实现：

```
$('#my-image')
    .slideUp('slow')
    .delay(1000)
    .slideDown('fast');
```

220 这种方法很有用，但是不够灵活，因为一旦设置，就无法取消延迟。

在介绍第9章之前，队列函数还有一个值得讨论的问题。

8.5.3 插入函数到特效队列

我们提到jQuery内部使用了一个名为`fx`的队列来排序要实现动画效果的函数。如果想在队列里穿插自己的函数，如何实现？既然知道了队列方法，就可以实现！

回头考虑列表8.7的例子——使用四个动画来让月亮围着地球旋转。想象在第二个动画之后修改月亮的背景色为黑色（向下移动的函数）。如果在第二步和第三步之间调用`css()`方法，代码如下：

```
var $moonImage = $('img[alt="moon"]');
$moonImage.animate({left: '+=256'}, 2500);
$moonImage.animate({top: '+=256'}, 2500);
$moonImage.css({backgroundColor: 'black'});
$moonImage.animate({left: '-=256'}, 2500);
$moonImage.animate({top: '-=256'}, 2500);
```

我们会感到失望，因为这会导致背景立即变黑，其实在第一个函数中就可以开始（记住`animate()`是非阻塞的）。思考下面修改的代码（**粗体**）：

```
var $moonImage = $('img[alt="moon"]');
$moonImage.animate({left: '+=256'}, 2500);
$moonImage.animate({top: '+=256'}, 2500);
$moonImage.queue('fx',
    function() {
        $(this)
            .css({backgroundColor: 'black'});
            .dequeue('fx');
    };
$moonImage.animate({left: '-=256'}, 2500);
$moonImage.animate({top: '-=256'}, 2500);
```

这里在一个函数里包装了调用`css()`方法，然后使用`queue()`把函数放到了`fx`队列中（我们忽略了队列名，因为默认是`fx`，这里澄清一下）。这会把`colorchanging`函数放到特效队列里，在第二个和第三个动画特效之间，它会作为执行动画特效函数链的一部分被执行。

但是注意！调用`css()`方法后，在`fx`队列上调用`dequeue()`方法。这绝对是必要的，以保持动

画队列的连续性。如果此时调用`dequeue()`方法失败，那么将会导致动画停顿，因为没有什么引起队列下一个动画的链式调用。未执行的动画将会保留在特效队列里，直到有什么引发其执行，或者页面卸载导致所有东西销毁。

除了这些改变之外，请思考在动画完成之后背景是什么颜色？因为没有执行其他的修改，所以它会保留黑色。这是我们希望避免的问题，所以应该恢复月亮的颜色。`animate()`方法的最后一个参数——回调函数，就是我们需要的。可以在回调函数里恢复最初的背景色（白色），代码如下：

```
$moonImage.animate({top: '-=256'}, 2500, function() {
    $(this).css({backgroundColor: 'white'});
});
```

也可以使用#FFFFFF 或等价的变量取代 white

如果想立即看到这些代码的效果，则可以打开代码文件`chapter-8/revolutions.2.html`，然后点击“Animate”按钮即可。

队列函数非常方便，虽然需要连续执行函数，但没有在异步回调函数里内嵌函数的开销和复杂度。今天有更高级的方法来处理这些问题（称为“callback hell”（回调地狱）），我们会在第13章介绍`Deferred`和`Promise`高级对象的时候讨论它。但这是另外一章的主题了。

8.6 总结

Summary

本章介绍了jQuery提供的动画效果，以及`animate()`方法，可以允许我们创建自定义动画。

`show()`方法和`hide()`方法，当无参调用时，会立即显示和隐藏元素，而没有任何动画效果。可以调用动画版的方法，通过传递参数来控制动画的速度，此方法提供了可选的回调函数，可以在动画完成时调用。`toggle()`方法会卷起元素，在显示和隐藏状态之间转换。

另外还有`fadeOut()`方法和`fadeIn()`方法，也可以通过调整元素的透明度来隐藏和显示元素。与`hide()`方法和`show()`方法相似，褪色效果也有一个`fadeToggle()`方法。另外一个此类型的方法叫`fadeTo()`，可以把元素的透明度修改为设置的值，而不需要从显示中删除元素。

最后三个内置的动画特效函数，通过调整元素的高度（`slideUp()`、`slideDown()`、`slideToggle()`）来删除或者显示选择的元素。

前一种方法介绍了`easing`概念。这个词用来描述动画处理的方式和每一帧变化的步骤。jQuery内核只提供两个`easing`函数以保持库的简单，且可以使用其他库或者插件来扩展，最有名的就是jQuery Easing插件和jQuery UI，可以获取许多新的`easing`函数。

此外，jQuery还允许我们使用`animate()`方法来自定义动画。通过使用此方法，可以调整接受数值的任何CSS样式属性，最常见的就是透明度、位置和元素的尺寸。

jQuery工具函数操作DOM

Beyond the DOM with jQuery utility functions

本章内容

- jQuery属性。
- 避免jQuery和其他库之间的冲突。
- 数组操作函数。
- 扩展和合并对象。
- 解析不同的格式。
- 动态加载新脚本。

到目前为止，我们花了大量章节介绍操作使用`$()`函数选择的DOM元素集合的jQuery方法。但是，你可能还记得，第1章中已经介绍了效用函数的概念——通过jQuery/`$`分配命名空间，但是不操作jQuery对象。这些函数可以被认为是最顶层的函数，只是它们被定义在`$`实例而不是Window对象上，把这些函数保留在全局范围之外。通常，这些函数操作的是JavaScript对象而不是DOM元素，或执行一些非对象相关的操作（例如Ajax请求或解析XML字符串）。

除函数外，jQuery提供了一些属性（有时称为标志），这些属性在jQuery/`$`命名空间内定义。其中一些属性只是供内部使用，但由于它们列举在jQuery官方网站的API文档里，所以有些插件也用到了它们。为了满足大家的好奇心，我们认为还是值得介绍的。

你可能会好奇为什么要等到本章才开始介绍这些函数和属性。有两个原因：

- 想要引导大家思考jQuery方法的使用，而不是低级的操作。
- 在页面中操作DOM元素，这些方法能帮助我们处理更多的工作，当编写自己的方法（以及其他扩展），而不是页面级的代码时，这些方法十分有用（在第12章将介绍如何扩展编写jQuery插件代码）。

本章不讨论处理Ajax的工具函数，会在第10章介绍。下面开始介绍提到的这些属性。

9.1 使用 jQuery 属性

Using the jQuery properties

jQuery为可用页面作者可用的一个新功能，不是通过方法或函数，而是作为属性定义在\$对象中。在过去，几个jQuery插件的作者一直依靠这些功能来开发其插件。在有些页面还可以看到这些功能，但是其中的有些功能已经过时，不再推荐使用。

jQuery 3:属性移除

jQuery 3已经删除了过时的文本内容(<https://api.jquery.com/context/>)、支持(<https://api.jquery.com/jQuery.support/>)和选择器(<https://api.jquery.com/selector/>)属性。如果你还在项目中使用它们，或者依赖于一个或多个插件，那么升级到jQuery 3会破坏你的代码的插件。

这些是可用的jQuery属性。

- \$.fx.off: 启用或禁用的影响。
- \$.fx.interval: 更改动画效果。
- \$.支持: 详细信息支持的功能(仅限于内部使用)。

出于好奇: \$.browser 属性

之前在jQuery 1.9版本中提供了一组属性给开发人员用于分支的代码(基于给定属性的值执行不同的操作)。当jQuery库被加载时设置，在任意ready处理器执行之前可用。它们被定义为\$.browser属性，这些可用的标志是msie、mozilla、webkit、safari及opera。

225

下面来看看这些属性，通过观察jQuery如何让你禁用启动动画。

9.1.1 禁用动画

有时，在一些网页中想要有条件地禁用动画，可能包括各种动画的效果。你可能会这样做，因为已经检测到平台或设备很难很好地处理这些问题，这或许是辅助功能的原因。例如，你可能想要在一些低配置移动设备上完全禁用动画效果，因为影响是缓慢的，可能为用户带来糟糕的使用体验。当发现在动画不利的环境中或不需要它们时，可以设置值\$.fx.off为true。

这不会抑制页面上使用的任何效果，它会简单地关闭这些动画效果。例如，淡入淡出效果会立即显示和隐藏元素、无干预的动画。同样，调用animate()方法将CSS属性设置为指定的最终值，没有对它们进行动画处理。

`$.fx.off` 标签, 以及第9.1.2节中讨论的`$.fx.interval`是一个读/写标志, 这意味着可以阅读以及设置其值。相反, `$.support`就要设置为只读。

9.1.2 改变动画速度

与其他执行动画的框架一样, jQuery有一个名为`$.fx.interval`的属性, 它可以设置该动画执行的速度。正如我们知道的那样, 动画有一系列步骤, 它们作为一个整体来创建特效。如果类比对学习有帮助, 大家可以把电影当做按照一定速度显示照片的过程。使用`$.fx.interval`属性就可以控制动画执行步骤的速度。

`$.fx.interval`属性为读/写属性, 且其值单位为毫秒, 其默认值为13。后者不是随机的, 但是它允许我们在流畅动画和CPU压力之间做出合理选择。

在高压动画驱动下, 可能想要尝试调整这个值来打造流畅的动画。可以设置`$.fx.interval`的值小于13来实现这个目标。修改速度在某些浏览器中会带来更好的效果, 但在低端设备或慢速浏览器引擎(例如, 那些旧版本的Internet Explorer)中就无法实现。在这些浏览器中, 不仅可能没有明显的优势, 还可能会带来糟糕的性能, 因为它给CPU太大的压力。

◀ 226

既然如此, 假设想要设置的值`$.fx.interval`为10, 则可以这样编写代码

```
$.fx.interval = 10;
```

同样, 可以设置为较低的值, 也可以再增大其值。这种变化可能变得方便, 如果正在处理一个网页, 则增加了CPU压力, 例如, 如果在同一时间运行几个动画, 或执行CPU密集型操作的同时运行一些动画。因为动画不如执行任务一样重要, 所以可以决定降低执行动画的速度来帮助CPU缓减压力。例如, 可以将值设置为100, 这样会导致10次/秒的刷新:

```
$.fx.interval = 100;
```

为了让大家看到`$.fx.interval`值控制动画播放效果的差别, 创建了一些演示代码, 可以在chapter-9/`$.fx.interval.html`文件中找到, 也可以在JS Bin (<http://jsbin.com/tevoy/edit?html,css,js,output>)中找到。

现在, 已经完成了动画处理的属性, 下面快速学习通过用户代理提供的环境信息的属性。

9.1.3 \$.support 属性

jQuery有一个名为`$.support`的属性, 它用于存储一些功能测试的结果, 对于库来说比较有意思。此属性允许jQuery在浏览器中决定支持哪些功能或不支持哪些功能, 而采取相应的操作。此属性仅供jQuery内部使用, 并且在jQuery 1.9版本中已被弃用, 所以我们强烈建议你不要使用它。一些属性, 现在或者将来将会公开, 包括`boxModels`、`cssFloat`、`html5Clone`、`cors`和`opacity`。

依赖\$.support对象不是一个好的主意，因为可能在任何时候未经通知的情况下删除其属性，当内部不再使用的时候。删除这些属性是为了提高库的负载的性能，因为它避免了要在浏览器上执行一些测试。本书中提到它，因为可能会使用旧的但很好的插件，它依赖于\$.support属性。

这个对象也是少有的不同jQuery分支之间存在差别的案例之一，目的也是不多情况下的其中一个。jQuery分支之间的区别：1.x分支比2.x分支有更多的属性，与兼容3.x相比，3.x有更多属性。例如，1.x有ownLast、inline-BlockNeedsLayout和deleteExpando，但2.x没有。

227

Modernizr 特征检测

如果项目需要基于浏览器支持的功能执行不同的方式，则应该采用一种称为功能检测（feature detection）的方法。代替检测用户使用的浏览器，然后试图确定浏览器是否支持该功能（称为浏览器检测（browser detection）的方法），功能检测需要我们直接检测功能的存在。

对于一个项目，需要测试几个功能，我们强烈鼓励使用这一特定目的而创建的外部库，最有名的和最常用的库是Modernizr(<http://modernizr.com/>)。此库用于检测提供原生支持的功能，源于HTML5和CSS3的规范。

对于Modernizr，至少有一个主要的浏览器实现的功能测试（测试一项功能是没有意义的，没有人支持，对吗？），但通常支持两个或更多。下面提供一个列表，Modernizr能做什么：

- 在几毫秒内超过40项功能的测试。
- 创建一个JavaScript对象(命名为Modernizr)，包含的测试结果为布尔型属性。
- 将类添加到html元素中用于描述功能的实现。
- 提供一个脚本加载程序，允许在旧的浏览器中使用polyfills回填功能。
- 允许在一些旧的浏览器中使用新的HTML5分节元素。

什么是填充工具？

填充工具是一段代码（或插件），规定开发人员期待的浏览器以自身方式提供技术。这个词于2010年由一位著名的JavaScript开发者和创始人Remy Sharp在founder of the Full Frontal conference上提出。可以在<http://remysharp.com/2010/10/08/what-is-a-polyfill/>中找到有关这个词的更多信息是如何以及为何在原Remy Sharp后创造，“什么是填充工具？”

本节已经学习了最后一个属性，现在准备继续前进并开始讨论 jQuery 的实用工具函数。

9.2 通过 jQuery 使用其他库

Using other libraries with jQuery

当在同一个页面使用其他库时，定义的全局名字\$通常是最大的争论和冲突点。如你所知，jQuery使用\$作为jQuery的简称，它可以使用jQuery暴露的每个功能。但是其他库，最有名的Prototype，也使用了\$名字。

jQuery提供了\$.noConflict()工具函数来释放\$标识符的控制权给其他库使用。这个函数的语法如下。

228

\$.noConflict() 函数语法

\$.noConflict(jQueryPropertyToo)

把\$的控制权释放给第三方库，允许在页面上同时使用jQuery和其他库。执行一个函数，jQuery功能调用是使用jQuery标识符（window对象的jQuery属性）而不是\$标识符。

jQuery标识符也可以通过给此函数传递true而放弃。

这个方法应该在引用jQuery库之后、添加第三方库之前调用。

参数

jqueryPropertyToo(Boolean) 如果设置为true，除了\$，jQuery标识符也会被放弃；否则保留。

返回

jQuery集合。

\$是jQuery的简称，所以所有的jQuery功能在调用\$.noConflict()之后仍然可用，通过window对象的jQuery属性进行访问。当然，也可以重新定义一个缩写标识符，以减少打字量。可以补偿这些损失，简单且好用的\$，通过定义新的不冲突的jQuery简称来实现，比如，

```
var $j = jQuery;
```

假设需要同时放弃\$和jQuery，则可以通过调用\$.noConflict()工具函数实现，把返回值保存在一个全局属性里：

```
window.$new = $.noConflict(true);
```

一旦语句执行，就可以使用\$new属性来调用jQuery方法了（例如\$new('p').find('a')）。

一种经常看到的设计模式，称为Immediately-Invoked Function Expression(IIFE，立即调用函数表达式)，包含创建环境，这个环境里\$标识符的作用范围是指jQuery对象（如果从没听过这种模式，或者需要复习，请阅读附录部分）。这个技巧在几个场景中常用到：扩展jQuery，尤其是被插件作者使用；模拟私有变量；处理闭包。对于插件作者来说，这非常有用，因为他们不能够确定页面开发者是否调用了\$.noConflict()，而且更确定的是，他们不能违背页面开发者的意愿来自己调用这个方法。

我们会在附录部分详细介绍这种设计模式。此刻，如果编写如下代码：

```
(function($) {  
    //函数体代码  
})(jQuery);
```

则不论Prototype或者其他外部函数是否定义了这个标识符，都可以在函数体内部安全地使用\$标识符。这非常酷，是不是？

我们做一个小测试来证明本节讨论的内容。测试第一部分，在列表9.1中检查HTML文档（chapter-9/overriding\$.test.1.html）。

列表 9.1 重写\$测试 1

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Overriding $ - Test 1</title>  
    <link rel="stylesheet" href="../css/main.css"/>  
  </head>  
  <body>  
    <button id="button-test">Click me!</button>  
  
    <script src="../js/jquery-1.11.3.min.js"></script>  
    <script>  
      $ = 'Hello world!';  
      try {  
        $('#button-test').on('click', function() {  
          alert('$ is an alias for jQuery');  
        });  
      } catch (ex) {  
        alert('$ has been replaced. The value is "' + $ + '"');  
      }  
    </script>  
  </body>  
</html>
```

① 使用自定义值重写 window.\$属性（字符串）

② 为页面按钮添加处理器

弹出成功消息 ③

弹出失败消息 ④

在这个例子中，导入jQuery，它定义了全局的jQuery及其简称\$。然后定义全局变量\$给一个字符串值①，以重写jQuery的定义。虽然可以使用简单的字符串来取代\$，但是它可以被其他库比如Prototype重新定义。

然后尝试给页面定义的按钮添加处理器②。在处理器内部，我们调用了JavaScript alert()函数，在屏幕上展示成功的消息③。如果出错，则会看到错误消息④。

在浏览器里加载页面，证明看到了如图9.1所示的错误消息。失败的原因是重新定义了\$标识符。而且，按钮没有触发任何操作，因为在添加处理器之前\$被重新分配了。

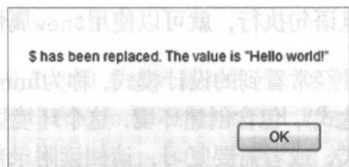


图 9.1 页面展示了\$已经被取代。值是“Hello world!”，因为它的定义已经起作用

现在只需要做小小的改动就可以安全使用\$。下面的代码展示了在try里做的修改。主要的修改代码用粗体字显示出来，以方便阅读。例子的完整代码可以在chapter-9/overriding\$.test.2.html文件里找到。

```
try {
  (function($) {
    $('#button-test').on('click', function() {
      alert('$ is an alias for jQuery');
    });
  })(jQuery);
} catch (ex) {
```

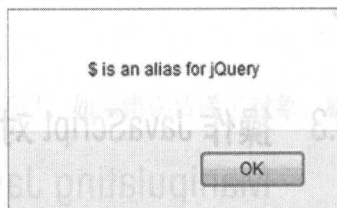


图 9.2 在处理器里设置的弹出框显示成功消息

唯一的修改就是，IIFE包装了附加处理器的代码，而且传递了window.jQuery属性给它。当加载这个版本的代码时，再点击按钮，就可以看到按钮正常工作了，显示的消息如图9.2所示。

修改上一个例子的代码，允许我们使用\$作为jQuery的简称，以保存最初的全局值\$（此时是字符串"Hello world!"）。现在看看如何恢复在jQuery调用\$.noConflict()方法之前\$的值。使用此方法的一个典型场景就是列表9.2，可以在chapter-9/\$.noConflict.html和JS Bin (<http://jsbin.com/bolok/edit?html,console>)中找到。

列表 9.2 使用\$.noConflict()方法

```
<!DOCTYPE html>
<html>
  <head>
    <title>Using $.noConflict()</title>
    <link rel="stylesheet" href="../css/main.css"/>
  </head>
  <body>
    <script>
      window.$ = {
        customLog: function(message) {
          console.log('The function says: ' + message);
        }
      };
    </script>
    <script src="../js/jquery-1.11.3.min.js"></script>
    <script>
      console.log('customLog: ' + ($.customLog === undefined));
      $.noConflict();
      console.log('customLog: ' + ($.customLog === undefined));
      $.customLog('Old value restored!');
    </script>
  </body>
</html>
```

导入 jQuery 库

恢复旧的\$值

创建一个库赋值给 window.\$

测试是否定义 \$.customLog 在控制台打印结果

使用\$.customLog() 在控制台打印消息

测试是否定义 \$.customLog 在控制台再次打印结果

在这个例子里，我们创建了一个自定义JavaScript库①，值包含一个方法customLog()。可以

导入任何使用\$符号的库,比如Prototype,以替代虚拟库,但是为了简单起见,没有添加真实的库。然后导入jQuery库②取代存储在\$中的库。接下来检查虚拟库,调用customLog()方法,发现没有定义(jQuery不包含此方法)③。

下一行代码里,我们恢复了\$的值,为了引入jQuery库之前的值,调用了jQuery的noConflict()函数④。最后,通过\$再次测试customLog()方法,打印消息到控制台上⑤。

现在,既然已经学习了如何使用jQuery来避免与其他库的冲突,是时候学习jQuery的工具函数了。

9.3 操作 JavaScript 对象和集合

Manipulating JavaScript objects and collections

jQuery主要的工具函数设计用来处理JavaScript对象,而不是DOM对象。通常,任意设计来操作DOM的方法都是作为jQuery方法定义的。虽然某些函数可以用来操作DOM元素——毕竟是JavaScript对象——但关注工具函数不是以DOM为中心的。

这些函数从处理简单的字符串操作和类型测试,到复杂的集合过滤、序列化表单的值,以及通过属性合并实现表单对象的继承。下面先从简单的函数开始介绍。

9.3.1 修剪字符串

令人费解的是,在ECMAScript 5之前,String没有任何方法可以删除开始和结尾部分的空白字符。这样的基本功能是所有其他绝大多数编程语言里String类的一部分,但是JavaScript却神秘地缺少这个函数,直到最近的一些版本。这意味着不能在IE9之前的浏览器里使用这个功能。

字符串修剪是JavaScript程序常见的需求;一个明显的例子是在数据验证期间。因为空格在屏幕上不可见,所以用户很容易在有效字符前后误输入一些额外的空白字符。在验证期间,需要剪切去掉多余的空格,而不是提醒用户自己修改这些看不到的字符。

在引入JavaScript原生的方法(String.prototype.trim())之前,为了支持旧的和新的浏览器,jQuery提供了\$.trim()工具函数。在背后,为了改进性能,这个方法在支持的浏览器里使用了原生的String.prototype.trim()方法。\$.trim()函数的定义如下。

\$.trim()函数语法

\$.trim(value)

删除参数字符串的头部和尾部的空白字符,返回结果。

这里定义的空白字符与其他任意匹配JavaScript正则表达式\s的一样,它不仅匹配空白字符,还包括表单换页、新的生产线、返回、制表符和垂直制表符,以及Unicode字符\u00A0。

参数

value(String) 要修剪的字符。原始值不会修改。

返回

修剪后的字符串。

使用此函数剪切文本字符的例子如下：

```
var trimmedString = $.trim($('#some-field').val());
```

要知道，函数会把传递的参数转换为等价的String类型值，所以，如果错误传递了对象，就会获得对象"[object Object]"。

现在来看一些操作数组和其他对象的函数。

9.3.2 迭代属性和集合

通常，我们有一些由其他组件组成的非标量值，需要迭代元素集合进行处理。这个元素容器可能是JavaScript数组（包含任意数量的JavaScript值，包括其他数组）或者JavaScript对象（包括属性）的实例，JavaScript语言提供了方法可以迭代这些元素。对于数组，可以使用for循环语法；对于对象，可以使用for...in循环来迭代它们的属性（其他结构都可以，但此刻先忽略它们）。

分别编写例子代码，如下：

```
var anArray = ['one', 'two', 'three'];
for (var i = 0; i < anArray.length; i++) {
    //使用 anArray[i]做一些工作
}
var anObject = {one: 1, two: 2, three: 3};
for (var prop in anObject) {
    //使用 prop 做一些工作
}
```

小菜一碟，但是有些人可能认为语法烦琐、复杂——最常见的批评就是针对for循环。

几年前，Array的forEach()方法已添加到了JavaScript上。不幸的是，作为后来添加的特效，一些浏览器特别是IE9之前的版本都不支持。此外，因为此方法属于Array对象，所以它无法用在其他对象上。jQuery解决了这个问题！

◀ 233

我们知道jQuery定义了each()方法，允许方便地迭代所有jQuery集合中的元素，而不需要for循环语法。对于数组，或者类数组对象，以及对象，jQuery提供了类似的工具函数\$.each()。

真正的好处是使用了相同的语法，无论是处理数组，还是对象的属性。此外，它也可以用到

IE6~IE8中，语法如下。

`$.each()`函数语法

`$.each(collection, callback)`

通用的迭代器函数，它可以用在对象和数组上。数组和类数组对象通过数字索引迭代，从0到length-1。其他对象通过命名属性迭代。

参数

`collection(Array|Object)` 要迭代的数组（类数组）对象，或者属性被迭代的对象。

`callback(Function)` 集合中每个元素调用的函数。如果集合是数组（或者类数组），则回调函数为每个数组元素调用；如果是对象，则回调函数为每个对象属性调用。回调函数的第一个参数是数组元素的索引，或者对象属性的名字。第二个参数是数组项目或者属性值。

函数上下文(this)也作为第二个参数传递。

返回

传递相同的集合。

这个统一的语法可以用来迭代数组、类数组或者对象。使用这个函数，可以重写之前的例子：

```
var anArray = ['one', 'two', 'three'];
$.each(anArray, function(i, value) {
    //做一些工作
});
var anObject = {one:1, two:2, three:3};
$.each(anObject, function(name, value) {
    //做一些工作
});
```

使用`$.each()`内联函数听起来不错，这个函数可使编写迭代器函数更加方便，或者简化分解循环里的逻辑代码到一个新函数里，如下所示：

```
$.each(anArray, someComplexFunction);
```

注意，当迭代集合数据时，可以通过返回`false`来终止循环函数。相反，返回`true`表示继续执行，相当于`continue`，这意味着函数立即停止，执行下一个函数。

jQuery 3:增加功能

jQuery 3引入了`for-of`循环迭代jQuery集合中DOM元素的功能，作为ECMAScript 6规范的一部分。幸亏有这个功能，可以编写如下代码：

```
var $divs = $('div');
for (var element of $divs) {
    //使用元素做一些工作
}
```

请注意，没有在`element`变量前放置美元符来强调这是一个DOM元素，而不是一个jQuery集合组成的元素。

注意：使用`$.each()`函数可能是从语法便捷的角度考虑问题，但是通常比使用旧的`for`语法慢。是否使用新的语法完全取决于你自己。

有时候需要迭代数组，然后选择元素到新数组中。虽然可以使用`$.each()`，让我们来看看jQuery如何更方便地处理这个问题。

9.3.3 过滤数组

对于处理大量数据的应用程序来说，遍历数组找到匹配某些条件的元素是非常常见的需求。也许需要过滤某个条件之上或者之下的元素或者匹配某个模式的元素。对于这种类型的过滤，jQuery提供了`$.grep()`工具函数。

`$.grep()`的函数名可能让我们相信这个函数使用了正则表达式，就像UNIX系统的`grep`命令。但是，`$.grep()`这个过滤器使用的并非正则表达式，它是一个回调函数，调用者提供用来决定数据值是否包含还是排除结果集。

没有东西阻止我们在回调函数里使用正则表达式来实现功能，但是他们的使用并非是自动化的。

函数的语法如下。

`$.grep()`函数语法

`$.grep(array, callback[, invert])`

遍历数组，并且为每个值执行回调函数。回调函数的返回值决定值是否保存到`$.grep()`函数返回的新数组中。如果忽略`invert`参数或者为`false`，则回调函数的值是`true`，导致数据被收集。如果`invert`是`true`，则回调函数的值是`false`，导致值被收集。初始元素不变。

参数

`array(Array)` 函数的返回值决定了当前值是否被收集。这个函数接收两个参数：当前值及值的索引。返回`true`导致当前值被搜集，除非`invert`参数为`true`，此时操作相反。

`invert(Boolean)` 一个可选值，如果为`true`，就反转正常操作的函数。

返回

收集值的新数组。

假设要过滤数组中大于100的值，则可以使用下面的代码实现：

```
var bigNumbers = $.grep(originalArray, function(value) {
    return value > 100;
});
```

传递给`$.grep()`的回调函数可以使用任意代码来处理包含的值。这个判断代码根据需要可以简单或者复杂。

尽管`$.grep()`函数没有直接使用正则表达式，但是JavaScript正则表达式在回调函数里确实是强大的工具，可以确定是否排除这个值。思考下面的数组例子，识别出哪些值不符合美国邮政编码规则（ZIP 代码）。

美国邮政编码由5个数组+可选的短线+4个数字组成。对应的正则表达式为`/^\d{5}(-\d{4})?$/`，所以可以使用下面的代码来过滤原始数组：

```
var badZips = $.grep(
    originalArray,
    function(value) {
        return value.match(/^\d{5}(-\d{4})?$/) !== null;
    },
    true
);
```

值得注意的是，这个例子中使用了String类的`match()`方法来确定值是否匹配模式，而且`$.grep()`的`invert`参数设置为`true`可用来排除匹配模式的值。

搜集数组的子元素并不是要执行的唯一操作。下面看看jQuery提供的另外一个工具函数。

9.3.4 转换数组

数据也许并非我们需要的格式。另外一个常见的频繁操作就是在以数据为中心的Web程序里，转换（translation）一个集合到另外一个集合。虽然可以通过编写for循环从一个数组创建一个新数组，或者从一个对象创建新数组，但是jQuery使用`$.map()`工具函数让这个工作变得更简单。

`$.map()`函数语法

`$.map(collection, callback)`

迭代传输的数组或者对象，为每个项目调用回调函数，在新的数组中收集函数的返回值。

参数

`collection(Array|Object)` 数组或者对象，它的值被转换到新的数据组中。

`callback(Function)` 函数的返回值被收集在新的数组中，作为`$.map()`结果集返回。这个函数接收两个参数：当前值和初始数组值的索引。如果传递的是对象，则第二个参数是属性的当前值。

返回

收集值的新数组。

下面看一个实战使用`$.map()`函数的小例子：


```
var oneBased = $.map(
  [0, 1, 2, 3, 4],
  function(value) {
    return value + 1;
  }
);
```

传入的数组被转换为下面的形式：

```
[1, 2, 3, 4, 5]
```

值得注意的一个重要行为就是，如果函数返回的是`null`或者`undefined`，结果就不会被收集进来。此时，结果数组将会比初始数组小，而且元素之间按照顺序一一对应的关系也丢失了。

现在看看一个稍微复杂的例子。想象有一个保存字符串的数组，可能是从表单字段里收集的，希望能表示数值。现在想把字符串数组转换成对应的数`Number`。因为没有办法保证字符串是有效的数字，所以需要多加注意。思考下面的代码，可以在文件`chapter-9/$.map.html`和JS Bin(<http://jsbin.com/zonopor/edit?html,js,console>)中找到：

237

```
var strings = ['1', '2', '3', '4', '5', '6'];
var values = $.map(strings, function(value) {
  var result = new Number(value);
  return isNaN(result) ? null : result;
});
```

从一个字符串数组开始，每个元素希望表示的是数值。但是有个输入错误，`5`表示成了`5`。代码通过检查`Number`实例来处理这种问题是否成功。如果转换失败，则返回值将会是`NaN`。但是比较有趣的是`NaN`，根据定义，它不等于任何其他东西，包括它本身！因此表达式`NaN===NaN`的值是`false`！

因为无法为`NaN`使用比较操作符（不是一个数（Not a Number）），所以JavaScript提供了`isNaN()`方法，它可以用来测试`string-to-numeric`（字符串到数值的转换）。

这个例子里，失败的时候返回`null`，以确保结果数组包含数值，而任何错误值省略。如果要收集所有的值，可以允许函数为坏的值，返回`NaN`。

`$.map()`的另外一种有用的行为就是它优雅地处理了从转换函数里返回的数组，并存入结果数组中。思考下面的代码：

```
var characters = $.map(
  ['this', 'that'],
  function(value) {
    return value.split('');
  }
);
```

这条语句转换一个字符串数组作为字符数组。执行代码后，变量`characters`的值如下：

```
['t', 'h', 'i', 's', 't', 'h', 'a', 't']
```

这是由于使用了JavaScript的`split()`方法，当传输空字符串作为分隔符时，会返回字符串的字符数组。这个数组作为转换函数的结果，合并到结果数组中。

238 jQuery对于数组的支持并没有结束。还有一系列有用的函数，用起来得心应手。

9.3.5 更有趣的 JavaScript 数组

你是否遇到要确定一个数组中是否包含某个特定的值，或者这个值在数组中的位置？如果是这样，则可以使用`$.inArray()`函数，其语法如下。

`$.inArray()`函数语法

`$.inArray(value, array[, fromIndex])`

返回第一个出现的传递值的位置索引。

参数

`value(Any)` 要查询的值。

`array(Array)` 要查询的数组。

`fromIndex(Number)` 数组中开始查询的位置。默认是0，则查询整个数组。

返回

数组中第一个出现值的位置索引，如果没有，则返回-1。

使用这个函数的简单例子如下：

```
var index = $.inArray(2, [1, 2, 3, 4, 5]);
```

指定索引的结果是1，赋值给`index`变量。

另外一个数组相关的函数，从类数组对象里创建JavaScript数组。思考下面的代码：

```
var images = document.getElementsByTagName('img');
```

这会使用`HTMLCollection`为变量`images`赋值所有页面上的图片元素。

处理这个问题，或者相似的对象会比较痛苦，因为缺少原生的JavaScript `Array`方法，像`sort()`和`indexOf()`。转换`HTMLCollection`为JavaScript数组会让工作变得简单。jQuery提供了此时需要的`$.makeArray`函数，其语法如下。

`$.makeArray()`函数语法

`$.makeArray(object)`

把类数组对象转换为JavaScript数组。

参数

`object(Object)` 要转换为数组的对象。

返回

JavaScript数组。

此功能适合在很少使用jQuery的代码中使用，它在内部处理了这种问题。在处理类数组的arguments参数对象时，此函数也有用武之地。想象有下面的函数，我们希望在内部排序参数：

```
function foo(a, b) {
    //排序参数
}
```

使用arguments类数组参数，可以一次获取所有参数。问题是arguments并非Array类型，所以不能这样编写代码：

```
function foo(a, b) {
    var sortedArgs = arguments.sort();
}
```

这段代码会抛出错误，因为arguments并不包含JavaScript的Array的sort()方法，此方法可以用来排序参数。在这种情况下\$.makeArray()可以帮助我们。可以把arguments转换为对应的数组，然后排序即可：

```
function foo(a, b) {
    var sortedArgs = $.makeArray(arguments).sort();
}
```

一旦修改完成，sortedArgs变量就会包含传递给foo()函数的参数数组。

现在假设有下面的代码：

```
var arr = $.makeArray({a: 1, b: 2});
```

一旦执行代码，arr将会包含一个元素组成的数组，它就是传递给\$.makeArray()的对象。

另外一个很少使用的函数，可能在处理jQuery之外构建数组的时候有用，它就是\$.unique()函数，其语法如下。

\$.unique()函数语法

\$.unique(array)

给定一个DOM元素的数组，返回初始数组的唯一元素数组，按照文档顺序排序。

参数

array(Array) 要处理的DOM元素数组。

返回

按照文档顺序排序的DOM元素的数组，包含初始数组的唯一元素。

该函数用于在jQuery范围之外创建的DOM元素的数组中使用。虽然许多人认为这个函数可以用于字符串数组或者数字数组，但是我们想强调的是\$.unique() 只能用于DOM元素数组。

在深入介绍下一个函数之前，我们来看一个使用\$.unique() 函数的例子。思考下面的标签代码：

```
<div class="black">foo</div>
<div class="red">bar</div>
<div class="black">baz</div>
<div class="red">don</div>
<div class="red">wow</div>
```

现在假设要查询所有包含样式black的<div>，作为DOM元素数组，然后把页面中的所有<div>存入集合，最后删除重复的元素。可以使用下面的代码实现这个需求（使用日志语句显示元素数量的不同）：

```
var blackDivs = $('.black').get();
console.log('The black divs are: ' + blackDivs.length);
var allDivs = blackDivs.concat($('.div').get());
console.log('The incremented divs are: ' + allDivs.length);
var uniqueDivs = $.unique(allDivs);
console.log('The unique divs are: ' + uniqueDivs.length);
```

加入要学习的例子代码，可以在文件chapter-9/\$.unique.html 和在线的JS Bin(<http://jsbin.com/borin/edit?html,js,console>)中找到。

jQuery 3:方法重命名

jQuery 3重命名\$.unique() 工具函数为\$.uniqueSort()，是为了说明此函数适合排序。除了这个修改，jQuery 3中仍然可以调用\$.unique()，不过它只是\$.uniqueSort() 的简称，但是已经过时了，后面的版本会删除这个函数。

现在假设要合并两个数组。jQuery提供了完成此任务的函数\$.merge()。

\$.merge()函数语法

\$.merge(array1, array2)

第二个数组的值合并到第一个数组，然后返回结果。第一个数组被修改后作为结果返回，第二个数组不变。

参数

array1(Array) 合并结果的数组。

array2(Array) 结果合并到第一个数组中。

返回

第一个数组，合并后的结果数组。

思考下面的代码：

```
var arr1 = [1, 2, 3, 4, 5];
var arr2 = [5, 6, 7, 8, 9];
var arr3 = $.merge(arr1, arr2);
```

此代码执行后，arr2是不变的，但arr1和arr3包含以下内容：

```
[1, 2, 3, 4, 5, 5, 6, 7, 8, 9]
```

注意，结果集里出现了两个5，因为\$.merge()工具函数不会删除重复值。

9.3.6 扩展对象

虽然知道JavaScript提供了一些特性，让它看起来像面向对象的语言一样，但是JavaScript并非纯面向对象的语言，因为有些特性不支持。其中一个重要的特性就是inheritance（继承，ECMAScript 6提供了支持）——新的类可以通过扩展现有类的定义实现。

JavaScript中一种模拟继承的模式是通过复制基类的属性到新对象中来扩展对象，可使用积累的功能扩展新对象。

通过复制实现扩展，在JavaScript代码中十分简单，但是这么多过程，jQuery提供了专门的扩展工具函数来解决这种需求：\$.extend()。正如将会在第12章里看到的关于jQuery插件，这个函数十分有用，不仅仅是扩展对象。它的语法如下。

\$.extend()函数语法

\$.extend([deep,] target, [source1, source2,...,sourceN])

通过传递的对象属性来扩展target对象。扩展后的对象可作为返回值。

参数

deep(Boolean) 一个标志，用于决定是深拷贝还是浅拷贝。如果忽略或者为false，则使用浅拷贝扩展。如果为true，则使用深拷贝进行扩展。

target(Object) 使用源元素属性来扩展目标对象。如果对象只提供参数，则它会作为源，jQuery对象作为目标。如果提供多个参数，则这个对象会使用新属性直接修改，然后作为结果返回。

与源元素属性名称相同的属性都会被源值重写。

242

source1 ... (Object) 一个或者多个对象，它们的属性会添加到target目标对象中。当提供多个源时，**sourceN** 同名的属性，后来的源覆盖前面同名的属性。

返回

扩展后的目标对象。

因为这个函数的灵活性，所以它的行为很有意思。几乎签名中的每个参数都是可选的，而且允许修改函数的功能。参数null或undefined被忽略。

如果只提供一个参数，则它不是作为目标，而是作为源。这种情况下，假设jQuery对象作为目标，属性会合并到jQuery对象中。

不值得合并包含比目标和源属性总数少的对象，即使它们都不一样。原因是\$.extend()会忽略值为undefined的属性。我们来看下这个函数的例子。设置了三个对象，一个目标、两个源对象，如下所示：

```
var target = {a: 1, b: 2, c: 3};
var source1 = {c: 4, d: 5, e: 6};
var source2 = {c: 7, e: 8, f: 9};
```

使用\$.extend()代码操作这些对象：

```
$.extend(target, source1, source2);
```

此代码把源对象内容合并到目标对象中。要测试这个代码，可以在chapter-9/\$.extend.test.1.html文件中找到并执行代码，结果会显示在页面上。在浏览器加载页面，结果如图9.3所示。

\$.extend() - Test 1

```
target (before) = {a: 1, b: 2, c: 3}
source1 = {c: 4, d: 5, e: 6}
source2 = {c: 7, e: 8, f: 9}
target (after) = {a: 1, b: 2, c: 7, d: 5, e: 8, f: 9}
```

图 9.3 \$.extend() 函数合并了多个源对象的属性，不包含重复值，而且按照顺序排列

所有source对象的属性都被合并到target对象中。但是注意下面几点：

- 所有包含名为c属性的对象。source1中的c替换了target对象中的c初始值，它又被source2中c的值取代。
- source1 和source2都包含名为e的属性。source2 中e的值覆盖了source1中合并到target里e的值，这也演示了后来列表中同名属性值会覆盖前面的同名属性值。

243

在继续学习之前，我们来看另外一个例子，正如作者几次面对的情况一样。假设你想合并两个对象，并且要保留两个的值，这意味着不想修改target目标对象的属性。为了执行这个操作，可以传递空对象作为目标，如下所示：

```
var mergedObject = $.extend({}, object1, object2);
```

通过传递空对象作为第一个参数，object1和object2都作为源处理；因此两个对象都不会被修改。

最后一个例子使用\$.extend()的第一个参数来演示如何执行深拷贝。假设有如下对象：

```
var target = {a: 1, b: 2};
var source1 = {b: {foo: 'bar'}, c: 3};
var source2 = {b: {hello: 'world'}, d: 4};
```

使用以下代码调用\$.extend()方法操作对象：

```
$.extend(true, target, source1, source2);
```

我们创建了可以重放这个例子的页面，便于大家学习，文件在chapter-9/\$.extend.test.2.html中。在浏览器里加载页面，结果如图9.4所示。

\$.extend() - Test 2

```
target (before) = {a: 1, b: 2}
source1 = {b: {foo: bar}, c: 3}
source2 = {b: {hello: world}, d: 4}
target (after) = {a: 1, b: {foo: bar, hello: world}, c: 3, d: 4}
```

图 9.4 演示\$.extend()函数合并内置对象的例子

这个函数非常有用。现在继续新的内容，因为还有一些工具函数要学习。

9.3.7 序列化参数

在动态、高交互性的应用程序中，这种需求非常常见。提交请求经常发生，而且这些请求是作为表单来提交的，浏览器会格式化请求的消息体来包含要提交的请求参数。其他时间，我们会作为URL来提交a元素的href属性。作为后者，我们的责任是创建和格式化查询字符串，包含要提交的请求参数。

服务端模板工具通常包含强大的机制来帮助我们创建URL，但是，当在客户端动态创建URL时，JavaScript并没有提供太多帮助。记住，不仅要正确配置(&)和(=)在查询字符串中的格式，还要确保名字和值被URL编码。虽然JavaScript提供了工具(`encodeURIComponent()`)，但是查询字符串的格式化还是需要我们来处理。

244

正如你可能期待的，jQuery提供了一个更加方便的工具：`$.param()`函数。

`$.param()`函数语法

`$.param(params[, traditional])`

序列化为适当的字符串传递信息，便于为查询字符串提交请求。传递的值可以是表单元素的数组、jQuery对象或JavaScript对象。查询字符串被格式化，而且每个名字和值被URI编码。

参数

`params(Array|jQuery|Object)` 要被序列化为查询字符串的值。如果传递的是数组或者jQuery对象，表单控件中表示的name/value（名/值）对会被提交到查询字符串中。

如果传递的是JavaScript对象，则对象的属性组成参数的名字和值。

`traditional(Boolean)` 可选标志，表示是否执行传统的影子序列化。它通常只影响包含内置对象的源对象。如果忽略，则默认是false。

返回

格式化后的查询字符串。

实战看下如何使用这个方法，思考下面的语句：

```
$.param({
  'a thing': 'it&s=value',
  'another thing': 'another value',
  'weird characters': ' !@#$$%^&*()_+= '
});
```

这里传递一个包含三个属性的对象给`$.param()`函数，为了URL传递的有效性，它的名字和值都包含要编码的字符串。这个函数调用的结果如下：

```
a+thing=it%26s%3Dvalue&another+thing=another+value&weird+characters= !%40%23%24%25%5E%26*()_%2B%3D
```

注意，查询字符串被正确格式化，而且名字和值中没有非数字和字母字符被编码。这可能导致字符串难以阅读，但是服务器代码能够正确处理这些字符串！

提醒：如果传递的是元素数组，或者包含元素的jQuery对象，而不是包含表单值，那么结果可能包含许多错误，如

undefined=&

245 在结果中，因为这个函数无法处理元素参数。

你可能认为这不是什么大问题，因为毕竟，如果只是表单元素，它们就可以被浏览器通过表单来提交，浏览器会处理所有这些工作。好吧，先记住这个问题。第10章中要介绍Ajax，到时你会看到表单元素并非一直被其表单提交。

但是这不会成为一个问题，因为会在后面看到jQuery提供了一种更高级别的方式（内部使用了这个工具函数），以更巧妙的方式处理这种问题。

现在来思考另外一个例子。想象页面中有下面的表单代码：

```
<form>
  <label for="name">Name:</label>
  <input id="name" name="name" value="Aurelio" />
  <label for="surname">Surname:</label>
  <input id="surname" name="surname" value="De Rosa" />
  <label for="address">Address:</label>
  <input id="address" name="address" value="Fake street 1, London, UK" />
</form>
```

如果用所有包含input元素的jQuery对象作为参数来调用\$.param()函数，代码如下：

```
$.param($('input'));
```

将会得到下面的结果字符串：

```
name=Aurelio&surname=De+Rosa&address=Fake+address+123%2C+London%2C+UK
```

这个例子演示了\$.param()如何与表单元素一起工作。但是关于此函数的讨论还没有结束。

序列化内置参数

经年累月被HTTP和HTML表单控件处理工作训练，Web开发者都会条件反射地想到序列化参数、查询字符串，还有一堆名/值对的列表。例如，想象一个表单，里面包含某人的性能和地址。此表单的查询参数或许包含字段firstname、lastname和city。序列化后的查询字符串可能是：

```
firstname=Yogi&lastname=Bear&streetaddress=123+Anywhere+Lane&city
=Austin&state=TX&postalcode=78701
```

序列化之前的数据结构可能如下：

```

{
  firstname: 'Yogi',
  lastname: 'Bear',
  streetaddress: '123 Anywhere Lane',
  city: 'Austin',
  state: 'TX',
  postalcode: '78701'
}

```

246

作为对象，不一定按照我们认为的方式来表示数据。从数据组织的角度来看，也许认为这个数据有两个主要的元素，即name和address，每个元素都有自己的属性，可能的代码组织结构如下：

```

{
  name: {
    first: 'Yogi',
    last: 'Bear'
  },
  address: {
    street: '123 Anywhere Lane',
    city: 'Austin',
    state: 'TX',
    postalcode: '78701'
  }
}

```

这个内置版本的元素，虽然比原始版本的逻辑结构更强，但是不容易转换为查询字符串。对吧？

通过使用习惯的标注方括号，这个结构可以表示为如下的查询字符串：

```

name[first]=Yogi&name[last]=Bear&address[street]=123+Anywhere+Lane&address[city]=Austin&address[state]=TX&address[postalcode]=78701

```

在这个标记中，子属性使用方括号保留数据结构。许多语言如PHP可以很方便地解码这些字符串。

这是一种聪明的选择，但并不是传统的JavaScript处理对象的方法。你可能期望有如下代码：

```

name=[object+Object]&address=[object+Object]

```

这没有什么作用。

幸运的是，jQuery可以处理内置参数，允许我们决定什么时候使用传统或者聪明的行为。我们要做的就是传递true来获取传统对象，传递false（或者忽略）来使用聪明的行为模式，作为\$.param()的第二个参数。

可以通过\$.param()试验页面来证明这个特性，例子在chapter-9/lab.\$.param.html中，结果如图9.5所示。

例子页面展示了\$.param() 如何序列化普通对象和内置对象，使用了聪明的算法模式及传统的算法模式。



继续操作这个试验页面，页面中设置了两个对象，所以可以立即启动测试\$.param()。我们也添加了编辑功能及新的属性，这样就可以使用不同的对象结构来进行序列化测试。

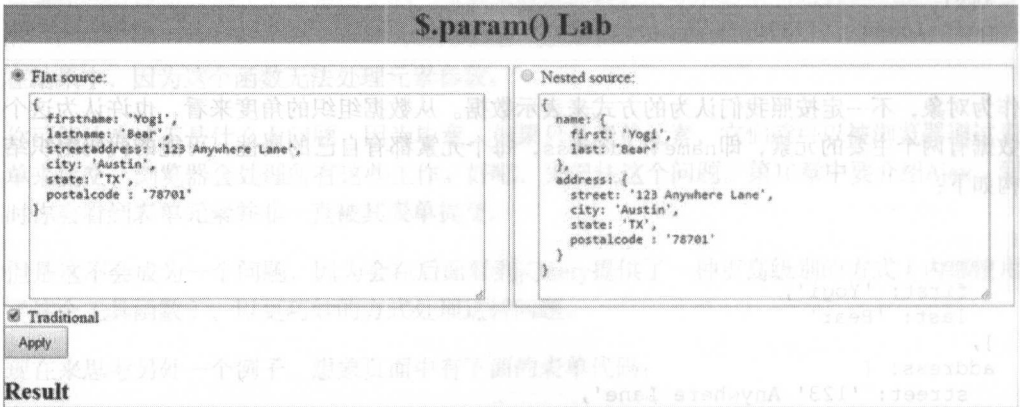


图 9.5 \$.param() 试验页面让我们看到了普通对象和内置对象如何序列化，分别使用了新的和传统的算法

9.3.8 测试对象

你可能已经注意到，许多jQuery方法和工具函数有多个参数列表；可选参数可以忽略，不需要的时候可以包含null值。以on()方法为例，它最常用的签名就是，

on(eventType[, selector][, data], handler)

如果没有传递选择器或者数据，那么可以简单地调用on()方法，处理器函数作为第二个参数，不需要方括号。jQuery通过测试参数的类型来处理这个问题，如果看到的只有两个参数，且函数作为第二个参数，那么它会把函数作为处理器，而不是作为选择器或者数据看待。

为不同的类型测试参数，包括是否是函数，如果要创建自己的函数或者方法，就会十分方便。因此，jQuery暴露了许多测试函数，如表9.1所示。

表 9.1 测试对象的 jQuery 工具函数

名 称	描 述
\$.isArray(param)	如果 param 是 JavaScript 数组，就返回 true(类数组对象如 jQuery 集合不会)；否则返回 false
\$.isEmptyObject(param)	如果 param 是 JavaScript 对象，不包含属性，包括从 prototype 继承任何属性，就返回 true；否则返回 false
\$.isFunction(param)	如果 param 是函数，则返回 true；否则返回 false

名 称	描 述
<code>\$.isNumeric(param)</code>	如果 param 表示数值, 则返回 true; 否则返回 false
<code>\$.isPlainObject(param)</code>	如果 param 表示用 via {} 或者 new Object() 创建 JavaScript 对象, 则返回 true; 否则返回 false
<code>\$.isWindow(param)</code>	如果 param 表示 window 对象, 则返回 true; 否则返回 false
<code>\$.isXMLDoc(param)</code>	如果 param 表示 XML 文档或者 XML 节点, 则返回 true; 否则返回 false

◀ 248

了解这些函数不错, 但是有一个实战的例子会更好。现在就开始介绍之!

假设需要一个函数, 它接收数组或者对象作为第一个参数, 而且乘以数组或者对象中的每个值, 乘数作为第二个参数。此外, 还需要指定一个函数作为乘法以后的处理函数, 它作为第三个参数。为了更有意思, 第二个参数 (叫 factor 因子)、第三个参数 (叫 customFunction) 都是可选参数。这意味着可以避免将它们作为一个参数。函数必须返回与第一个参数类型相同的新对象, 而不能修改后者。

根据描述, 函数的前面可以表示如下:

```
multiplier(collection[, factor][, customFunction])
```

使用表 9.1 中列举的方法, 不需要其他代码就可以处理所有这些情况。实现代码如列表 9.3 所示。也可以在 chapter-9/testing.functions.html 和 JS Bin (<http://jsbin.com/lolub/edit?js,console>) 中找到源代码进行测试。

列表 9.3 测试工具函数

```
function multiplier(collection, factor, customFunction) {
  function calc(value) {
    return $.isFunction(factor) ?
      factor(value) :
      $.isFunction(customFunction) ?
        customFunction(value * factor) :
        value * factor;
  }

  var result = null;

  if (factor === undefined && customFunction === undefined) {
    factor = 1;
  }

  if ($.isArray(collection)) {
    result = $.map(collection, function(value) {
      if ($.isNumeric(value)) {
        return calc(value);
      }
    });
  }
}
```

① 定义 calc() 支持函数, 包含核心计算代码

② 如果未定义第二个和第三个函数, 则设置因子为 1

③ 如果处理数组, 则使用 \$.map() 函数为每个元素调用 calc()

◀ 249

```

    } else if ($.isPlainObject(collection)) {
        result = {};
        for(var prop in collection) {
            if ($.isNumeric(collection[prop])) {
                result[prop] = calc(collection[prop]);
            }
        }
    }

    return result;
}

```

如果处理对象，使用 for 循环为每个对象值调用 calc()

⑤ 返回计算结果

这段代码列举了本章中介绍的许多函数，所以会详细介绍。

在函数的第一部分定义了 calc() 支持函数的核心计算①。它用于处理 multiplier() 函数的参数，且根据第二个参数值执行不同的操作。如果第二个和第三个参数未定义，则设置因子为1，然后开始计算。如果集合是数组类型，则函数使用 \$.map() 工具函数为每个数组元素调用 calc()，但是只在它是数值的时候②。如果集合的类型是 Object，则函数在对象的值上使用 for 循环来调用 calc()，但是只在它是值的时候③。最后返回结果④。结果的数据类型取决于第一个参数的类型（数组或者对象）。如果既不是数组也不是对象，就会返回 null。

本节介绍的函数允许测试变量中包含的值是否是特定的类型（对象、数组或者函数等）。但是，如果想知道变量的类型信息呢？

查看值的类型

jQuery 提供了另外一个函数 \$.type() 来处理这个问题。函数的语法如下。

\$.type() 函数语法

\$.type(param)

确定值的类型。

参数

param(Any) 要测试的值。

返回

描述值的类型信息的字符串。

为了更好地理解调用这个函数的结果，假设有下面的代码：

250 > \$.type(3);

这种情况下，会得到如下结果：

"number"

845 > 如果代码如下：

\$.type([1, 2, 3]);

将会获得下面的结果：

```
"array"
```

相比通常在JavaScript中测试类型的方式，这是一个重大的不同，当深入研究插件时会派上用场。事实上，如果支持如下测试：

```
if (typeof [1, 2, 3] === 'array')
```

那么结果是false，因为typeof [1, 2, 3]的返回值是"object"。

已经完整学习了工具函数处理数据类型的知识，现在看看一些允许处理不同类型字符串的函数。

9.3.9 解析函数

jQuery提供了一系列工具函数来解析不同的数据格式，从JSON到XML，再到HTML。新的浏览器提供了对象JSON来处理JSON格式的数据。这个对象包含parse()方法，正如名字所示，解析JSON字符串。确切地说，新的浏览器，通常指的是支持这个功能的浏览器。幸运的是，jQuery又来学习雷锋了，提供了\$.parseJSON()工具函数，其语法如下。

\$.parseJSON()函数语法

\$.parseJSON(json)

解析传入的JSON字符串，返回等价的结果。

参数

json(String) 要解析的JSON字符串。

返回

等价的JSON字符串。

我们已经多次看到，当浏览器支持原生方法的时候，jQuery直接使用原生方法，而且这个函数也不例外。如果浏览器支持JSON.parse()，jQuery将会使用它；否则会使用JavaScript工具来执行等价的操作。而且，jQuery尽可能地改进了操作的性能。

JSON字符串必须具备良好的格式，而且这个规则就是良好格式的JSON比JavaScript表达式标记更严格。例如，所有属性名必须用双引号字符分隔，即使它们形成有效的标识符。无效的JSON (251)将会导致错误。参见官方网站 (<http://www.json.org>) 关于良好格式的JSON定义。

JSON并不是Web上交换信息的唯一格式。另外一种常见的格式就是XML。jQuery允许我们解析XML字符串，通过\$.parseXML()工具函数转换为等价的XML文档，语法如下。

\$.parseXML()函数语法

\$.parseXML(xml)

解析XML字符串为XML文档。

参数

xml(String) 要解析的XML字符串。

返回

包含数据的XML文档。

XML文档易于使用和遍历，因为jQuery支持它们，所以可以通过\$.parseXML() 传递对象给jQuery，然后使用目前为止学习的方法。听起来很疯狂？不要担心。此时，将会展示一个试验页面，以帮助大家练习这些概念。

最后一个属于此类别的方法就是\$.parseHTML()。它使用原生的DOM元素创建函数来把包含HTML的字符串转换为DOM元素集合，然后使用jQuery方法来操作这些元素。

此函数的语法如下。

\$.parseHTML()函数语法

\$.parseHTML(html[, context][, keepScripts])

解析字符串为DOM节点数组。

参数

html(String) 要解析的HTML字符串。

context(Element) 可选参数用来作为HTML代码生成的上下文。如果不指定，或者传递null或undefined，默认值就是当前的document。

keepScripts(Boolean) 如果为true，则函数会在HTML字符串中包含脚本。默认值是false。

返回

字符串生成的DOM元素的数组。

这个函数描述中，我们指定了默认的keepScripts参数值为false。这个决定的背后有着非常重要的安全考虑。当从外部源获取HTML字符串时，如果包含脚本，则可能会对网站造成恶意的攻击。

252 值得一提的是，在绝大部分环境中，尽管已经去除了script元素，但仍然可以实施攻击，所以应该确保不会有从未信任的资源（比如URL或者cookie）输入。作为不通过script元素实施的攻击，思考可以通过img元素的onerror属性执行的操作。



刚刚讨论的三个工具函数可以给我们创建另外一个试验页面的机会，可以在chapter-9/lab.parsing.html文件中看到例子源代码，如图9.6所示。

这个试验页面包含三个预制的代码块。第一个是可以使用点符号查询JSON对象，可以通过\$.parseJSON() 函数把原始文本转换为JavaScript对象。

第二个代码块允许查询XML和HTML代码，方式与第2章看到的试验页面一样。这时只显示有多少个元素被选中。此部分的试验页面使用了\$.parseXML() 和\$.parseHTML()。

注意，可以修改这三个代码块，因为把它们放到了textarea元素里。因此，可以测试自己的JSON对象、XML代码或者HTML代码。打开试验页面，现在开始练习吧。

一旦熟悉了这些方法，就可以来学习其他的工具函数了。

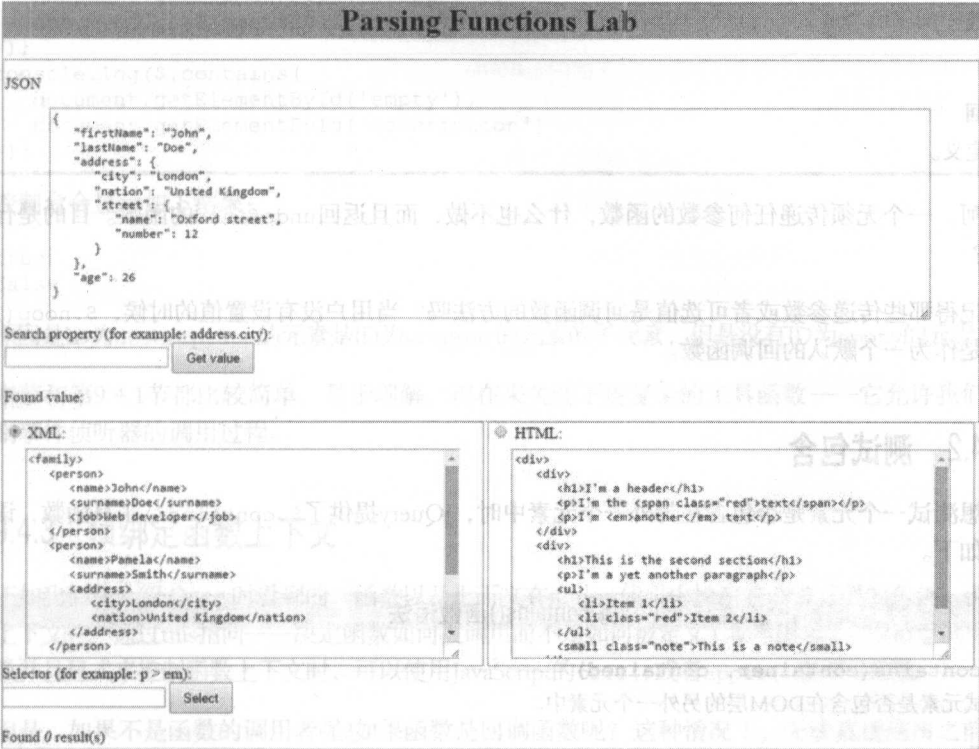


图 9.6 解析函数试验页面允许我们测试 jQuery 函数处理三种支持的格式：JSON、XML、HTML 253

9.4 其他工具函数

Miscellaneous utility functions

本节将会介绍一系列工具函数，这些函数基本上自成一体。现在就从一个无所事事的函数开始吧。

9.4.1 什么也不做

jQuery定义了一个什么也不做的工具函数。我们猜测这个函数可以有如下的名字 `$.uselessFunctionThatDoesNothing()`，但是这个太长了，所以它的真实名字是 `$.noop()`。其语法如下。

`$.noop()` 函数语法

`$.noop()`

什么也不做。

参数

无。

返回

未定义。

呵呵，一个无须传递任何参数的函数，什么也不做，而且返回 `undefined` 的函数。目的是什么？

还记得那些传递参数或者可选值是回调函数的方法吗？当用户没有设置值的时候，`$.noop()` 就是作为一个默认的回调函数。

9.4.2 测试包含

当想测试一个元素是否包含在另外一个元素中时，jQuery提供了 `$.contains()` 工具函数，语法如下。

`$.contains()` 函数语法

`$.contains(container, contained)`

测试元素是否包含在DOM层的另外一个元素中。

参数

`container(Element)` 要测试的DOM容器元素。

`contained(Element)` 被测试的DOM元素。

返回

如果包含的元素包含在容器内，则返回 `true`；否则返回 `false`。

嗨，等一下！这是不是听起来很熟悉？实际上，我们只谈第2章里讨论的 `has()` 方法，这两个函数有惊人的相似之处。

这个函数在jQuery内部使用非常频繁，当有引用的DOM元素来测试时非常有用，而且没有必

要花费开销来创建jQuery集合。

在实际代码中查看这个方法的使用情况，HTML代码如下：

```
<div id="wrapper">
  <p id="description">Some text</p>
</div>
<div id="empty"></div>
```

如果执行下面的两段代码：

```
console.log($.contains(
  document.getElementById('wrapper'),
  document.getElementById('description')
));
console.log($.contains(
  document.getElementById('empty'),
  document.getElementById('description')
));
```

控制台会显示如下结果：

```
true
false
```

原因是ID为description的元素是ID为wrapper的元素的子元素，但是没有ID为empty的元素。

本节和第9.4.1节都比较简单，易于理解。现在来关注下更复杂的工具函数——它允许我们控制事件侦听器的调用过程。

9.4.3 预绑定函数上下文

正如我们在介绍jQuery时看到的，函数以及上下文在jQuery代码中扮演着非常重要的角色。函数上下文——通过this指向——决定函数如何被调用而不是如何被定义（参考附录）。当希望调用函数并且显式来控制函数上下文时，可以使用JavaScript的call()或者apply()来调用函数。

但是，如果不是函数的调用者呢？如果函数是回调函数呢？这种情况下，无法直接使用之前提及的方法来影响函数上下文的设置。

jQuery提供了工具函数，通过它可以预绑定一个对象给函数，这样当函数调用的时候，绑定对象就会变成函数的上下文。这个工具函数叫\$.proxy()，它的语法如下。

\$.proxy()函数语法

\$.proxy(function, proxy[, argument, ..., argument])

\$.proxy(proxy, property[, argument, ..., argument])

接受一个函数，然后返回一个新的包含特定上下文的函数。

参数

function(Function) 要修改上下文的函数。

proxy(Object) 设置为上下文的对象(this)。

argument(Any) 传递给函数参数function的引用的参数。

property(String) 要修改上下文的函数的名字(proxy对象的属性)。

返回

新的函数, 它的上下文设置为proxy对象。

在浏览器里打开例子文件chapter-9/\$.proxy.html, 结果如图9.7所示。

在这个例子页面中, 有个“Test (测试)”按钮, 它的ID为test-button, 把它存储到变量中, 代码如下:

```
var $button = $('#test-button');
```

当点击“Normal”单选按钮时, “Test”按钮就会建立一个处理器和容器:

```
$button.click(customLog);
```

customLog() 处理器在屏幕上显示了函数上下文的ID (this引用的):

```
function customLog() {
    $('#log').prepend(
        '<li>' + this.id + '</li>'
    );
}
```

当点击按钮时, 希望事件处理器所在的元素“Test”按钮作为函数的上下文。点击“Test”按钮的结果如图9.8所示。

当点击“Proxied”单选按钮时, 事件处理器建立如下:

```
$button.click($.proxy(customLog, $('#control-panel').get(0)));
```

和之前建立的处理器一样, 除了处理器函数是通过\$.proxy() 工具函数传递的, 还预绑定了一个对象给处理器。这种情况下, 可以绑定ID为control-panel的元素。要绑定的对象不一定是元素——实际上, 大部分情况都不是。这个例子选择的元素是为了便于通过ID进行区分。

现在, 当点击“Test”按钮时, 获取的结果如图9.9所示, 它证明了函数上下文已经通过

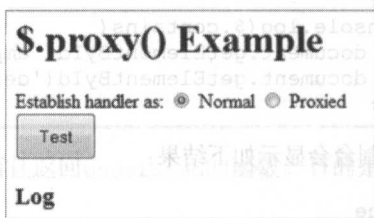


图 9.7 \$.proxy 例子代码帮助我们理解正常与 proxied 回调之间的差别

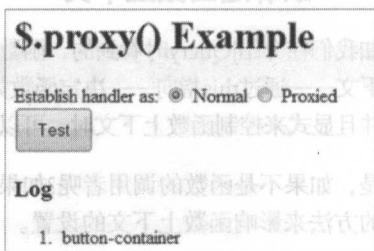


图 9.8 当使用正常处理器时, \$.proxy 例子的结果

“\$.proxy()” 函数强制设置为特定的对象。

这个功能在为回调函数提供数据的时候非常有用，它可能无法通过闭包或者其他方式正常访问数据。

\$.proxy() 最常见的使用场景就是，当我们想把一个对象的方法作为处理器绑定，而且对象作为处理器函数的上下文时。思考下面例子代码中的对象：

```
var obj = {  
  id: 'obj',  
  hello: function() { alert('Hi there! I am  
    ' + this.id); }  
};
```

如果通过obj.hello()调用hello()方法，则函数上下文(this)就是obj。但是，如果使用如下代码来建立事件处理器，

```
$(whatever).click(obj.hello);
```

我们发现，函数上下文不是obj，而是当前冒泡事件的元素。如果处理器依赖于obj，那就悲剧了。可以使用\$.proxy()工具函数来强制设置obj作为函数的上下文，代码如下：

```
$(whatever).click($.proxy(obj.hello, obj));
```

当然，也可以使用第二个参数，作用是一样的，代码如下：

```
$(whatever).click($.proxy(obj, 'hello'));
```

obj就是函数要设置的上下文对象，“hello”字符串表示属于obj对象的方法，它的上下文将会被修改掉。

但是要知道，这种方法导致的结果就是，无法知道当前事件冒泡传播的元素——传入的新对象会作为函数新的上下文。

9.4.4 评估表达式

虽然很多开发者不推荐使用eval()，但是也有有用的时候（看一下解析函数试验页面的例子）。<257>

问题是eval()在当前上下文里执行。当编写插件和其他可重用脚本时，想确保评估发生在全局上下文里。现在就引入\$.globalEval()工具函数，语法如下。

\$.globalEval()函数语法

\$.globalEval(code)

在全局上下文里评估传入的JavaScript代码。

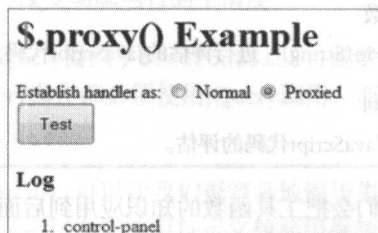


图 9.9 这个例子展示了预绑定对象给“Test”按钮的点击事件处理器

参数

code(String) 进行评估的JavaScript代码。

返回

对JavaScript代码的评估。

我们会把工具函数的知识应用到后面编写jQuery插件的章节中，但是也可能在其他情况下使用这些工具函数。

9.4.5 抛出异常

某些情况下，我们希望在函数或者插件中抛出异常。例如，如果开发者传递了一个意外的参数，则希望抛出一个错误。因此，jQuery提供了工具函数\$.error()，语法如下。

\$.error()函数语法

\$.error(string)

接受一个字符串参数，然后抛出一个包含此字符串的异常。

参数

string(String) 要发出的错误信息。

返回

未定义。

这个方法主要被插件开发者用于系统重写并提供更好的错误消息显示体验。此函数的简单例子如下：

```
function isPrime(number) {  
    if (typeof number !== 'number') {  
        $.error('The argument provided is not a number');  
    }  
    //在这里编写其余代码  
}
```

258 当讨论如何为jQuery开发插件的时候，会再次使用这个函数，但那是新一章的内容。

9.5 总结

Summary

本章深入学习了jQuery提供操作对象的其他工具函数，包括排序函数、标志函数、jQuery顶级（简称\$）直接定义的函数。

首先,我们学习了处理动画的标志。设置\$.fx.off可允许我们在网站上禁用动画,这样的变化会立即发生。然后介绍了\$.fx.interval,这个标志可改变动画运行的平滑度。

考虑到有些页面开发者需要与jQuery一起使用其他库,jQuery提供了\$.noConflict()函数,它允许其他库使用\$简称。在调用此函数后,所有的jQuery操作都必须使用jQuery调用,而不是\$。

jQuery还提供了一系列函数来处理数组中的数据。\$.each()可以让我们很容易地遍历集合元素;\$.grep()允许根据过滤条件来创建新的数组;\$.map()允许使用自定义转换根据源数据来生成对应的新的数组。

不仅可以使jQuery的\$.isArray()来测试某个值是否在数组中,而且可以测试某个值是不是数组。也可以使用\$.isFunction()来测试函数的真假,或者使用\$.type()工具函数来检查对象类型。

另外一些函数允许处理不同格式的数据。其中\$.parseJSON()和\$.parseXML()用于解析Web中最常见的两种数据格式信息:JSON和XML。剩下的\$.parseHTML()用来处理HTML标签代码。

除此之外,还有一些小众的函数,也可以使用jQuery的\$.extend()函数来合并对象。这个函数允许合并任意数量对象的属性到目标对象中。最后,\$.proxy()函数允许在代码中修改函数的上下文。

工具箱里有了这些额外的工具,就可以使用这些工具函数来执行Ajax请求了。

259

使用Ajax与服务器交互

Talk to the server with Ajax

本章内容

- Ajax简介。
- 从服务器加载预定义格式的HTML。
- 发送GET和POST请求。
- 细粒度控制请求消息。
- 设置默认的Ajax属性。
- 处理Ajax事件。

Ajax是众多改变互联网Web发展前景的技术之一。它的异步请求服务器而无须重新加载整个页面的能力，就产生了许多新的用户交互范式，而且让DOM脚本应用成为可能。

在微软公司发明Ajax技术的几年后，许多技术活动相继举办，让Ajax成为Web开发社区的共识。非微软公司的浏览器实现了一种标准化版本的技术——XMLHttpRequest (XHR)对象；Google公司开始使用XHR；而且在2005年，Adaptive Path公司的Jesse James Garrett 发明了词语Ajax（Asynchronous JavaScript and XML，异步JavaScript和XML）。

好像大家都在等待该技术有一个朗朗上口的名字，Web开发人员突然大量使用Ajax，而且它已经成为开发DOM脚本应用程序的主要方式之一。

本章将简要介绍Ajax（如果你已经熟悉Ajax，则可以直接跳到第10.2节），然后介绍jQuery如何使用Ajax技术。下面先从Ajax技术的概念开始吧。

10.1 复习 Ajax

Brushing up on Ajax

虽然本节要快速复习Ajax技术，但不是要做一个完整的Ajax手册或者Ajax指南。如果你完全不熟悉Ajax（或者更糟，认为我们正在谈论的是洗洁精或希腊神话英雄），那么希望你能通过一些公共的Ajax学习资源来提前熟悉下该技术。

有些人可能对词语Ajax存在争议，该词语被用来向服务器发送请求而不需要完全刷新用户页

面（比如通过提交请求给隐藏的iframe元素），但是绝大部分人还是把它与XMLHttpRequest (XHR)或者Microsoft XMLHTTP ActiveX控件关联到一起。整个Ajax请求的处理过程，每次执行一个步骤，如图10.1所示。

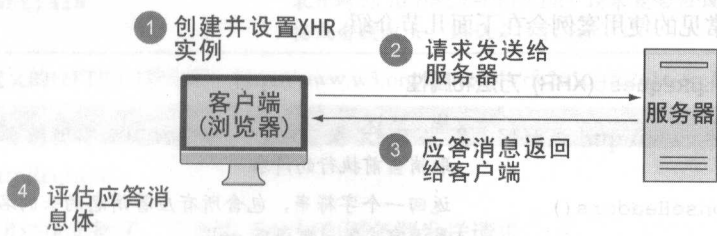


图 10.1 Ajax 请求的生命周期，从客户端到服务器，再到客户端

下面看看这些对象如何生成请求，现在从创建XHR对象开始。

10.1.1 创建 XHR 实例

在完美世界中，计算机代码可以运行在所有常见的浏览器中，但是，正如我们了解的，不可能生活在这样的世界里。这些差别对于Ajax来说，没有什么不同。通过JavaScript XHR对象发送异步请求有一个标准的方式，而且旧的IE浏览器可以使用ActiveX控件。IE7有一个包装的接口，而IE6需要特别处理代码。

注意：jQuery Ajax 实现——本章余下的内容会介绍这个——在可用的时候使用ActiveX对象。

这对于我们来说是一个好消息！为 Ajax 需求使用 jQuery，我们知道它已经封装了最好的方法。如果不需要支持旧的 IE 浏览器，那么开发工作会简单很多。

261

一旦创建，这些代码设置、初始化和响应请求对浏览器来说都是相对独立的，而且对于特定的浏览器，创建XHR实例非常简单。问题是对于不同的浏览器，XHR有不同的实现方式，需要为当前浏览器使用恰当的方法来创建对象实例。

但是，不是依赖于检测用户使用的是什么浏览器，而是确定哪条使用路径，下面将使用第9章介绍的技术功能检测（feature detection）。列表10.1的代码展示了典型的使用这种技术创建XHR实例的过程。

列表 10.1 功能检测可以让我们在多种浏览器里使用 Ajax

```
var xhr;
if (window.ActiveXObject) {
    xhr = new ActiveXObject('Microsoft.XMLHTTP');
} else if (window.XMLHttpRequest) {
    xhr = new XMLHttpRequest();
}
```

测试是否支持 ActiveX

测试是否支持 XHR

```

} else {
    throw new Error('Ajax is not supported by this browser');
}

```

如果不支持 Ajax, 则抛出一个错误

一旦创建, XHR实例具备一系列跨各种浏览器支持的属性和方法。这些属性和方法如表10.2所示, 而且最常见的使用案例会在下面几节介绍。

表 10.1 XMLHttpRequest(XHR) 方法和属性

方 法	描 述
abort()	取消当前执行的请求
getAllResponseHeaders()	返回一个字符串, 包含所有应答消息的头的名字和值, 如果没有应答, 则返回 null
getResponseHeader(name)	返回指定名称的文本值, 如果没有应答或者没有该属性, 则返回 null
open(method, url[, async[, username[, password]]])	设置 HTTP 方法 (GET 或者 POST)、请求的 URL 地址。可选择, 基于容易验证, 可设置请求为同步模式或者异步模式, 也可以为请求设置用户名和密码
overrideMimeType(mime)	设置 mime 应答的 Content-Type 头
send([content])	初始化请求。可选参数 content 提供请求体。如果是 GET 或 HEAD 方法, 则忽略 content
setRequestHeader(name, value)	使用指定的名字和值来设置请求头
属 性	描 述
onreadystatechange	当请求状态改变时, 执行事件处理器
readyState	一个整数值, 表示当前请求的状态: 0=UNSENT 1=OPENED 2=HEADERS_RECEIVED 3=LOADING 4=DONE
response	根据 responseType 的应答消息体
responseText	返回的应答消息体
responseType	可以设置应答消息类型。它的值可以是 "" (空字符串)、arraybuffer、blob、document、json 或者文本
responseXML	如果消息体内容为 XML, 就会从消息体内容创建 XML DOM
status	服务器返回的应答消息状态码。例如, 200 表示成功, 404 表示未找到。参考 HTTP 规范 ^a 里的所有状态码
statusText	应答消息返回的状态文本消息
timeout	在强制终止前请求可以耗费的毫秒时间, 默认是 0, 它表示没有超时
ontimeout	当请求超时时调用的事件处理器

属 性	描 述
upload	通过为 upload 添加事件处理器可以跟踪上传过程
withCredentials	表示跨站 Access-Control 请求是否应该使用 cookie 凭据或者授权头。默认是 false

a. RFC 2616定义的HTTP 1.1状态码: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10>。

注意: 要想获得权威的介绍, 可以查看 XHR 规范, 网址为 <http://www.w3.org/TR/XMLHttpRequest/>。

263

现在可以创建XHR实例了, 下面看看如何向服务器发送请求。

10.1.2 初始化请求

在向服务器发送请求之前, 需要按照以下步骤来操作。

- (1) 指定HTTP方法 (例如POST 或者GET)。
- (2) 提供服务器资源的URL地址。
- (3) 让XHR实例知道如何通知你它的进度。
- (4) 为POST这样的请求设置消息体内容。

通过调用XHR对象的open()方法来设置前两个参数, 代码如下:

```
xhr.open('GET', '/some/resource/url');
```

注意, 这个方法并不会把请求发送给服务器, 它只是设置URL和HTTP动词。open()也可以接受第三个布尔参数, 指定请求是异步还是同步 (默认是true, 表示异步)。几乎没有理由使用同步模式 (意味着不需要回调函数); 毕竟, 异步请求的本性才是使用它的根本原因。

在第三步, 必须提供一种方式给XHR实例, 以通知我们发生了什么。通过赋值回调函数给XHR对象的onreadystatechange属性即可。这个函数被称为准备状态处理器 (ready state handler), 被XHR实例在不同状态阶段调用。通过查看XHR其他属性的设置, 可以发现请求出了问题。第10.1.3节将会介绍典型的准备状态处理器是如何操作的。

最后一步, 初始化请求状态, 为诸如POST的请求设置消息体, 并发送给服务器。这两步都是通过send()方法完成的。对于GET或者HEAD请求, 它没有消息体, 无须传递消息体参数, 代码如下:

```
xhr.send();
```

当请求参数传递给其他请求类型时, 传递给send()的字符串必须是正确的格式 (查询字符串格式 (query string format), 名字和值都被正确地进行URI编码。URI编码超出了本节的范畴

(jQuery会处理这些工作)，但是，如果你好奇心很强，在JavaScript里，可以使用 `encodeURIComponent()` 函数编码。

这种调用的一个例子如下：

```
xhr.send('a=1&b=2&c=3');
```

264 现在来看看准备状态处理器是干什么的。

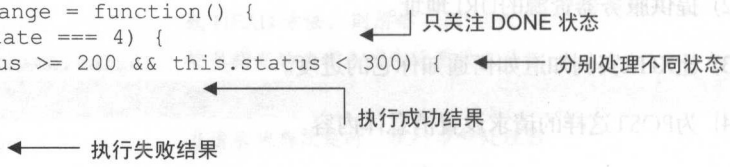
10.1.3 跟踪进度

XHR实例可以通过准备状态处理器 (ready state handler) 通知我们它的进度。这个处理器可以通过把函数赋值给XHR实例的 `onreadystatechange` 属性实现。

一旦请求通过 `send()` 方法初始化，当请求状态不停转换的时候，回调函数就会被多次调用。当前的请求状态可以通过 `readyState` 属性的数字代码来获取 (属性值介绍见表10.1)。这很好，但是，通常只对请求什么时候完成、是否成功或者失败感兴趣。我们经常看到，处理器的编写代码方式如列表10.2所示。

列表 10.2 编写的准备状态处理器代码，只处理 DONE (完成) 状态

```
xhr.onreadystatechange = function() {  
    if (this.readyState === 4) {  
        if (this.status >= 200 && this.status < 300) {  
            //成功  
        } else {  
            //问题  
        }  
    }  
}
```



这段代码忽略了其他状态，只处理DONE状态 (完成的状态值4)，而且一旦检测，就通过检查 `status` 属性来判断请求成功或者失败。HTTP规范定义了从200~299的成功状态码，以及其余的300以上的不同类型的状态码，表示失败或重定向等状态。

现在来看看如何处理完成请求的应答消息。

10.1.4 获取应答消息

一旦准备状态处理器确定 `readyState` 完成、请求成功，就可以从XHR实例获取应答消息的消息体。

尽管绰号是Ajax (x表示XML)，但是应答消息体的内存可以是任意的文本格式，不只局限在XML上。Ajax请求的应答消息可以是纯文本、HTML代码，或者使用JSON表示的任意数据。

关于它的数据格式，应答消息体可以通过XHR实例的 `responseText` 属性获得 (假设请求已经成功完成)。如果应答消息通过内容消息头指定MIME类型为 `text/xml` 或 `application/xml`，或

者以+xml结尾的MIME类型,那么应答消息体就会作为XML解析。结果DOM可以在responseXML属性里获取。JavaScript (以及jQuery本身,使用了它的选择器API)可以处理XML DOM。

◀ 265

此刻,你或许想回顾一下图10.1中的整个过程。在这个简短的Ajax复习章节中,我们已经指出了网页开发者使用Ajax需要处理的痛点。

- 实例化XHR对象,需要浏览器专门的代码。
- 准备状态处理器需要过滤一堆不感兴趣的状态转换。
- 根据内容格式,应答消息体需要不同的处理方式。

本章剩下的内容将会介绍如何使用jQuery Ajax和工具函数来简化页面编程。jQuery Ajax API提供了许多选择,而且会从一些最简单和常用的工具开始。

10.2 加载内容到元素中

Loading content into elements

可能Ajax最常见的用途就是从服务器抓取数据,然后显示到页面的某个位置的元素中。内存可以是HTML代码片段,也可以是原生的文本内容,都可以作为目标元素的子元素进行填充。

设置例子

与本书之前介绍的例子代码不同,本章的例子代码需要Web服务器来处理Ajax请求。因为这个服务端内部运行的知识超出了本书的范畴,所以会跳过这一部分。

我们将要使用的服务端代码是用PHP开发的,所以你的服务器应该允许它运行。假设已经配置完成,无论什么操作系统,这里以一个工具列表开始处理:

- Windows用户:<http://www.wampserver.com/en>。
- Mac用户:<https://www.mamp.info>。

如果使用的是其他语言,比如Java或者ASP.NET,则应该很容易转换为需要的代码,这很简单。如果决定把页面转换为Java或者ASP.NET,就需要配置对应的Web服务器,以支持该框架,例如Tomcat和IIS。这里是一些有用的资源:¹

- 对于Windows和Linux用户Tomcat:<https://tomcat.apache.org/tomcat-7.0-doc/setup.html>。
- 对于Mac用户Tomcat:<http://serverfault.com/questions/183496/how-do-istart-apache-tomcat-at-boot-on-mac-os-x>。
- IIS的Windows用户:<http://www.iis.net/learn/install/installing-iis-7/installingiis-on-windows-vista-and-windows-7>。

◀ 266

¹ 对于Windows 7、8、10的系统,在个人计算机上安装IIS很简单,也方便开发。可以在控制面板⇒程序和功能⇒打开关闭Windows功能⇒找到IIS并勾选,即可完成安装。——译者注

假设要从服务器端抓取名为some-resource的HTML资源，然后作为ID为elem的<div>的内容显示。最后，再来看看不使用jQuery如何实现这个功能。

使用本章早期介绍的内容，可以编写如列表10.3所示的代码。这个例子完整的HTML代码可以在文件chapter-10/listing.10.3.html中找到。

注意：再强调一次，必须在 Web 服务器里运行这个例子——不能直接在浏览器里打开这个例子文件——URL 应该是 http://localhost:8080/chapter-10/listing.10.3.html。如果使用 Apache，则可以忽略端口设置；如果是 Tomcat，则可以保留这个端口。在本章后面的内容里，会使用[:8080]来表示是否需要端口，但是一定不要用方括号括起来。

列表 10.3 使用原生 XHP 来获取包含 HTML 的数据

```
var xhr;
if (window.ActiveXObject) {
    xhr = new ActiveXObject('Microsoft.XMLHTTP');
} else if (window.XMLHttpRequest) {
    xhr = new XMLHttpRequest();
} else {
    throw new Error('Ajax is not supported by this browser');
}
xhr.onreadystatechange = function() {
    if (this.readyState === 4) {
        if (this.status >= 200 && this.status < 300) {
            document.getElementById('elem').innerHTML = this.responseText;
        }
    }
}
xhr.open('GET', 'some-resource');
xhr.send();
```

虽然这里没有太难的代码，只有几行代码（17行）。等价的jQuery代码如下：

```
$('#elem').load('some-resource');
```

我们打赌并知道，你宁愿编写和维护这个代码！

现在来详细介绍这个代码中使用的jQuery方法。

10.2.1 使用 jQuery 加载内容

这个简单的jQuery代码最后使用最常见也是最基本的jQuery Ajax方法来从服务端加载内容：load()。此方法的完整语法介绍如下。

load()方法语法

```
load(url[, data][, callback])
```

向指定的URL地址执行Ajax请求，传递可选数据参数。可以为请求完成指定调用的回调函数，且DOM被修改。返回的文本取代所有匹配的元素。

参数

`url(String)` 请求要发送给服务端资源的URL地址，通过选择可以修改（下面解释）。

`data(String|Object|Array)` 指定作为请求参数的数据。这个参数可以是查询字符串或者应答消息体，可以序列化属性的对象，或者指定name/value(名/值)对的数组对象。

如果指定了对象或者数组，则请求要使用POST方式。如果忽略或者指定字符串，则默认使用GET方式。

`callback(Function)` 在应答数据加载到匹配的元素以后调用的回调函数。传递这个函数的参数是应答文本、状态字符串（通常是"success"）及jqXHR实例（后面介绍）。

这个函数会为jQuery集中的每个元素执行调用，并使用之前设置的目标元素的上下文(this)。

返回

jQuery集合。

在这个方法中我们引入了一个新的对象jqXHR，它是jQuery XMLHttpRequest的简称，是XMLHttpRequest (XHR) 对象的超集。它包含responseText 和 responseXML属性，以及getResponseHeader()方法。而且它还实现了Promise接口，这会在第13章介绍。

虽然使用起来很简单，但是这个方法还有一些重要的细节。例如，当使用data参数来提供请求参数，且参数是对象时，请求使用的是HTTP POST方式；否则使用GET方式。如果要使用参数发起GET请求，则可以把参数包含到URL的查询字符串中。但是要知道，这样做要确保URL格式的正确性，且每个参数要被URI编码。JavaScript的encodeURIComponent()方法就是为了处理这种问题的，或者可以使用jQuery \$.param()工具函数，第9章中已做了介绍。

有时在把内容注入一个或者多个元素中后需要执行某些操作。假设要做一个网站，不停轮询服务器，更新伦敦地铁的状态，而且每次获取信息后都会在页面显示更新的信息。因此，在每个回调完成之后，需要不停地请求服务器。满足这个需求的简单例子实现如下：

```
var updates = 1;
function pollInfo() {
    $('#container').load(
        '/check-updates',
        function(responseText, textStatus, jqXHR) {
            if (textStatus === 'success') {
                $('#status-update').text('Data updated. Update #' + updates);
                updates++;
            }
            setTimeout(pollInfo, 1000);
        }
    );
}
pollInfo();
```

268

1000 毫秒后执行 pollInfo()函数

第一次调用 pollInfo()函数

绝大部分时间，使用load()方法会把jQuery对象中包含的应答消息内容注入其他元素中，但是有时我们希望过滤应答的内容。jQuery允许我们设置过滤器来筛选要注入元素中的应答内

容。可以通过在URL后加一个空格，然后添加选择器实现。

例如，要过滤应答消息，只有<div>元素才可以注入，可以编写如下代码：

```
$('.inject-me').load('/some-resource div');
```

选择器可以再复杂一些，这意味着可以编写如下语句：

```
$('.inject-me').load('/some-resource div .some-class a');
```

此时，jQuery会查询所有匹配选择器的元素。

当要在请求消息里附带数据时，有时要自定义数据，但通常是要从用户输入的表单元素里获取输入数据。

正如我们期望的，jQuery提供了一些帮助方法来完成这项处理工作。

从数据中序列化

如果发送的请求数据来自于表单控件，则jQuery提供转换查询字符串的serialize()方法语法如下。

serialize()方法语法

serialize()

为集合中的表单元素创建正确格式和编码的查询字符串。

参数

无。

返回

269 格式化的查询字符串。

serialize()方法足够灵活，它只从匹配元素集合的表单元素中获取信息，而且只是那些认为成功（successful）的元素。成功（successful）指的是根据HTML规范定义的表单提交规则¹。控件，比如没有选择的勾选框、单选按钮、无选择的下拉框及禁用的控件，都不会作为成功的控件，都不会参与表单提交，所以它们会被serialize()方法直接忽略。

序列化数据并发送给服务器是jQuery一项不错的扩展功能，但不幸的是，这个库并没有提供相应的反操作：反序列化（deserialize）。

反序列化根据查询字符串的值来修改表单字段的状态。在复杂的查询表单中，要存储一些查询条件时比较有用。此时，可以把字段的值保存到cookie或者数据库里，这样用户就可以点击按钮，然后恢复这些值，而不需要重新填写这些表单。

¹ HTML 4.01 规范，参见第 17.13.2 节“Successful controls”，网址为：<http://www.w3.org/TR/html401/interact/forms.html#h-17.13.2>。

当jQuery无法提供解决方案时,充满激情的社区搞定了此事。这也是为什么jQuery强大的原因。像之前描述的场景,可以使用jQuery插件jQuery.deserialize(<https://github.com/kflorenc/jquery-deserialize>)来解决。

使用 jQuery.deserialize 反序列化

要使用jQuery.deserialize方法,首先要在页面添加引用。引用方法在本书中介绍过很多次了——在jQuery库引用的<script>标签里添加链接地址,代码如下:

```
<script href="path/to/jquery.js"></script>
<script href="path/to/jquery.deserialize.js"></script>
```

插件设置完成,假设有下面的表单:

```
<form id="my-form">
  <input name="name" />
  <input name="surname" />
</form>
<button id="btn">Auto fill</button>
```

从前面的表单序列化可得到如下的查询字符串:

```
name=Aurelio&surname=De+Rosa
```

有了这个前提条件,当用户点击“btn”按钮时,若想要自动填充表单,那么可以编写如下代码:

```
$('#btn').click(function() {
  $('#my-form').deserialize('name=Aurelio&surname=De+Rosa');
});
```

如果希望把表单数据放到数组中(与查询字符串对应),则可使用jQuery提供的serializeArray()方法,它的语法如下。

serializeArray()方法语法

serializeArray()

收集成功的表单控件的值到数组对象中。

参数

无。

返回

数组形式的表单数据。

通过serializeArray()获得的数组由键/值对象组成,每个包含name属性和value属性,对应的是表单控件的名字和值。注意这个格式也适用于为load()方法传递参数。

下面使用load()方法来处理一些常见的真实世界的问题，许多Web开发者应该都遇到过。

10.2.2 加载动态 HTML 代码

在业务应用中，特别是电商网站，经常需要从服务端抓取实时数据来更新用户最新的信息。毕竟，我们不希望误导用户买到缺货但是想要的商品，对吧？

本节会从开发页面开始，整个网站贯穿本章内容。这个页面是名为The Boot Closet(靴子商店)的虚拟公司网站的一个网页。与其他在线零售商固定的产品目录不同，这个库存和盘货都是流动的，取决于老板当天的销售以及卖出的商品。始终显示最新的信息，这一点非常重要！

开始编写页面（会忽略导航以及其他模板，专注本节内容），需要展示给用户一个下拉框，包括当前所有靴子的样式。当选择一种样式时，还要显示样式的详细信息。初始页面如图10.2所示。



图 10.2 网上商店的初始页面，有一个允许用户点击的简单下拉框

在页面第一次加载后，包含样式的下拉框显示出来。当没有选择样式时，会显示：“- choose a style -”（选择一种样式）。这时会提示用户选择下拉框，而且当用户选择一种样式后，下面是要做的工作：

- 在下拉框下面的区域显示该样式的详细信息。
- 删除“- choose a style -”选项；一旦用户选择了样式，默认选项就不起作用了。

下面从定义页面结构的HTML标签代码开始：

```
<body>
  <div id="banner"></div>

  <h1>Choose your boots</h1>
  <div>
    <div id="selections-pane">
      <label for="boot-chooser-control">Boot style:</label>
      <select id="boot-chooser-control" name="model"></select>
    </div>
    <div id="product-detail-pane"></div>
  </div>
</body>
```

① 包含选择空间

② 产品详细信息

代码很少是吧？正如所预料的，把样式定义放入外部的CSS文件里（这里没有显示），坚持“低

调的JavaScript”原则，HTML标签里没有行为代码。

标签里最有意思的代码就是容器❶，它包含select元素，允许用户选择靴子的样式；而另外一个容器❷会显示该样式的详细信息。

注意，靴子样式的控件需要在用户选择之前添加选项元素。现在给页面添加必要的行为。首先要做的就是使用Ajax获取最新的靴子样式数据，然后添加给下拉框。

272

注意：绝大部分情况下，初始值都会在服务端处理，在发送给用户浏览器之前完成。

这意味着如果用户禁用 JavaScript 或者不能执行 JavaScript 代码，则仍然可以使用这些页面。有需要通过 Ajax 获取数据的情况，但是这里仅仅是为了演示学习的目的。

为了给靴子样式控件添加选项，需要使用load()方法：

```
$('#boot-chooser-control').load('actions/fetch-boot-style-options.php');
```

是不是很简单？代码唯一复杂的地方就是URL部分，但不是所有的URL都这么长、这么复杂，它指定了PHP服务端网页的请求。

使用Ajax的一个好处就是，它是独立于服务端技术的。发送HTTP请求，有时需要参数数据，而且服务端需要返回期望的结果，可以忽略服务端的是Java、ASP.NET、Ruby、PHP或者旧的CGI接口。

特殊情况下，希望服务端返回特殊的HTML代码，包括靴子样式的选项——期望是从仓库数据库中返回。模拟的服务端代码返回了如下应答消息：

```
<option value="">- choose a style </option>
<option value="7177382">Caterpillar Tradesman Work Boot</option>
<option value="7269643">Caterpillar Logger Boot</option>
<option value="7332058">Chippewa 9" Briar Waterproof Bison Boot</option>
<option value="7141832">Chippewa 17" Engineer Boot</option>
<option value="7141833">Chippewa 17" Snakeproof Boot</option>
<option value="7173656">Chippewa 11" Engineer Boot</option>
<option value="7141922">Chippewa Harness Boot</option>
<option value="7141730">Danner Foreman Pro Work Boot</option>
<option value="7257914">Danner Grouse GTX Boot</option>
```

这个结果注入了select元素，然后页面就得到了一个正常的下拉框控件。

下面的工作就是为下拉框编程，以响应用户的选择，这是我们余下的任务。以下代码稍微复杂了一些：

```
$('#boot-chooser-control').change(function(event) {
    $('#product-detail-pane').load(
        'actions/fetch-product-details.php',
        {
            model: $(event.target).val()
        },
        function() {
            // 获取并显示产品信息
        }
    );
});
```

❶ 创建 change 事件处理器

❷ 获取并显示产品信息

```
function() {
    $('[value=""]', event.target).remove(); ❸ 删除占位符
}
);
```

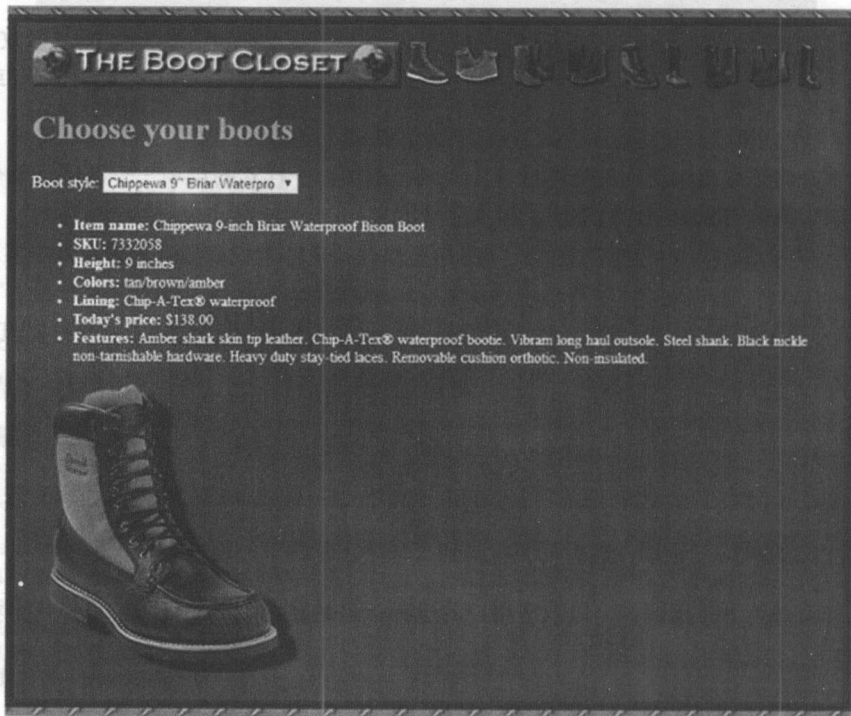
273

在这段代码里，我们选择了靴子样式下拉框，绑定了事件change处理器❶。在事件处理器里，当用户选择新的下拉框的值时就会调用此代码，可以使用jQuery的val()方法来获取当前选项的值。在这个例子中，目标元素就是触发事件的元素。

在product-detail-pane元素上，使用load()方法❷向服务器页面actions/fetch-product-details.php发送请求。对于这个页面，通过使用只包含model属性的对象来传递靴子样式。最后，在load()的回调函数里删除占位符选项❸。

在用户选择可用的样式以后，页面会出现如图10.3所示的结果。

最显著的操作就是使用了load()方法来快速从服务器获取样式HTML代码，然后注入现有的DOM元素中。这个方法十分方便，而且十分适用于服务端模板化的Web应用程序。



274

图 10.3 服务端资源返回了格式化的 HTML 代码来显示靴子的详细信息

列表10.4展示了Boot Closet（靴子商店）的网页完成代码，可以在文件http://localhost[:8080]/chapter-10/phase.1.html里找到。随着内容的深入，我们会在本章后面继续往这个网页里添加更多的功能。

列表 10.4 Boot Closet (靴子商店) 电商网页第一阶段的代码

```
<!DOCTYPE html>
<html>
  <head>
    <title>The Boot Closet - Phase 1</title>
    <link rel="stylesheet" href="../css/main.css" />
    <link rel="stylesheet" href="../css/bootcloset.css">
  </head>
  <body>
    <div id="banner"></div>

    <h1>Choose your boots</h1>
    <div>
      <div id="selections-pane">
        <label for="boot-chooser-control">Boot style:</label>
        <select id="boot-chooser-control" name="model"></select>
      </div>
      <div id="product-detail-pane"></div>
    </div>

    <script src="../js/jquery-1.11.3.min.js"></script>
    <script>
      $('#boot-chooser-control')
        .load('actions/fetch-boot-style-options.php')
        .change(function(event) {
          $('#product-detail-pane').load(
            'actions/fetch-product-details.php',
            {
              model: $(event.target).val()
            },
            function() {
              $('[value=""]', event.target).remove();
            }
          );
        });
    </script>
  </body>
</html>
```

`load()`方法在从服务端获取HTML代码并插入其他元素时，非常有用。但是，许多时候，我们希望更多地控制Ajax请求或者需要对返回的应答消息体做更多的处理工作。

下面继续学习jQuery提供的处理更多复杂问题的功能。

◀ 275

10.3 发送 GET 和 POST 请求

Making GET and POST requests

`load()`方法是发送GET请求还是POST请求，取决于请求传递的参数数据，但是有时我们希望对于使用的HTTP方法有更多控制。为什么要这样？因为可能是服务器的需要。

Web开发者通常采用GET方法和POST方法快速开发，而不需要关注HTTP协议如何封装这些

请求方法。实际上，每个方法的定义如下。

- GET请求——用于表示幂等（idempotent）操作；相同的GET请求，一次又一次，应该返回同样的结果（假设没有其他因素改变服务器状态）。
- POST请求——可以是非幂等的；发送给服务器的数据可以用来改变应用程序的状态模型，例如，添加、更新数据库记录或者删除服务器上的信息。

GET请求应该用来获取数据，正如其名字的含义一样。它有时也需要向服务器发送一些请求参数，例如，用样式数来查询颜色信息。但是，当要发送数据给服务器来修改数据时，就应该使用POST请求。

警告：这不仅仅是理论上的。浏览器会根据使用的 HTTP 方法来决定缓存策略，GET 是主推的缓存方式。使用正确的 HTTP 方法可确保我们不会误导浏览器或者服务器错误理解请求的目的。这仅仅是 RESTful 原则的一个方面，其他的 HTTP 方法，如 PUT 和 DELETE 也可以发挥作用。但是，我们的目标，限制在 GET 和 POST 方法的讨论上。

记住这些，重新看看第一阶段实现的例子——靴子商店（The Boot Closet，见列表10.4），你会发现我们之前做错了！因为当提供一个对象哈希作为data参数时，发送了一个POST请求，实际上可以发送GET请求。如果查看Chrome浏览器的开发者工具（Developer Tools）日志（见图10.4），当使用Chrome浏览器显示页面时，会看到第二个请求，当选择样式下拉框时，实际是一个POST请求。

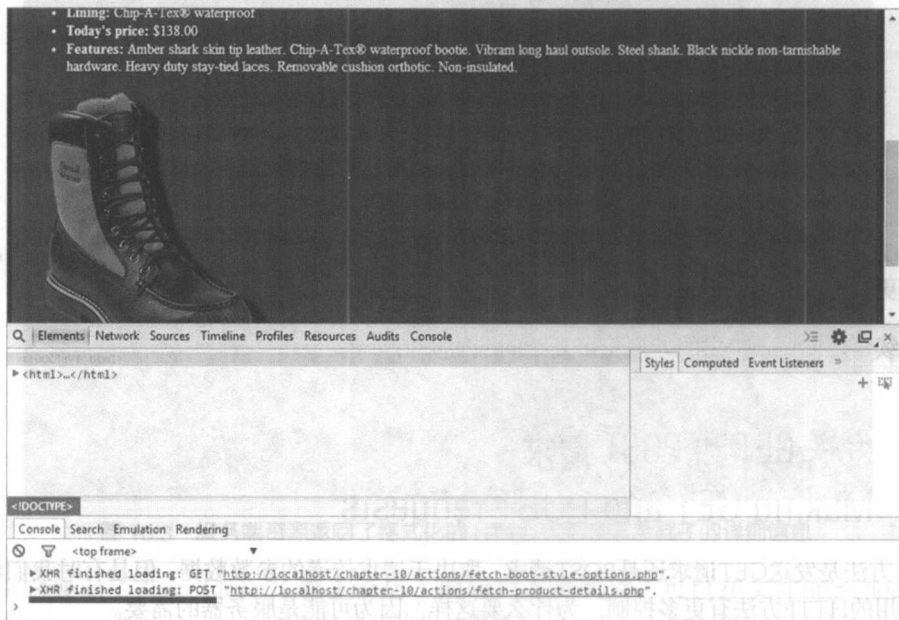


图 10.4 Chrome 浏览器控制台展示了发送的 POST 请求，实际应该是 GET 请求

注意：只有启用记录日志 XMLHttpRequests 功能，控制台看到的结果才与图片一样。

如果不想启用这个选项，那么可以看一下 Network（网络）选项卡。

这是问题吗？当然这取决于你自己，但是，如果想使用合适的HTTP方式，那么应该使用GET方式而不是POST获取数据。

◀ 276

开发工具

没有调试工具来开发DOM脚本化的应用程序，就好像带着焊接手套弹钢琴一样。为什么要这样折磨自己？

使用不同的浏览器，有不同的方式来检查代码。所有新的浏览器都内置了这种工具，可能名字不太一样，例如，Chrome浏览器里叫Developer Tools（开发者工具）(<https://developer.chrome.com/devtools/>)，而IE里叫F12 developer tools（F12开发者工具）([http://msdn.microsoft.com/en-us/library/bg182326\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/bg182326(v=vs.85).aspx))。

Firefox浏览器也有自己的内置工具，但是开发者同行使用的是Firebug(<http://getfirebug.com>)插件。这个工具不仅允许我们监测JavaScript控制台，还允许我们监测动态DOM元素、CSS、脚本，以及其他页面开发设计的问题。

◀ 277

和现在的需求紧密相关的一个功能就是，记录Ajax请求和应答的日志信息。

可以使用字符串来指定请求信息而不是对象（后面再讨论），但是现在，我们看看如何使用jQuery来初始化Ajax请求。

10.3.1 使用 GET 获取数据

jQuery提供了几种发送GET请求的方式，不像load()，这些工具不是作为jQuery方法而是作为函数封装。这就是第9章提到但是还没有讲解的函数。

当想从服务端获取数据，且决定如何处理这些数据(而不是让load()方法将其设置为一个或多个内容元素)时，可以使用\$.get()工具函数，其语法如下。

\$.get()函数语法

\$.get(url[, data][, callback][, dataType])

使用指定的URL和查询字符串参数向服务器发送GET请求获取数据。

参数

url(String) 使用GET方法访问的服务器URL地址。如果为空字符串，则请求会在调用的时候发送到当前地址。

data(String|Object) 查询字符串中传递的参数数据。参数是可选的，可以是字符串，也可以是对象，对象的属性和值会序列化为查询字符串。

callback(Function) 当请求成功完成时调用的函数。应答消息体作为回调函数的参数,根据dataType进行解析,状态字符串作为第二个参数。第三个参数包含对于jqXHR实例的引用。在回调函数内,上下文(this)表示调用设置的Ajax请求对象。如果设置了dataType,则这个参数是必须的。假设不需要函数,那么可以传递null或者\$.noop()作为占位符。

dataType(String) 可选参数,指定应答消息体的结构,可以是下列数据类型:html、text、xml、json、script或jsonp。在xml、json、script或html中,默认值根据应答消息获取。参考本章后面面对\$.ajax()的详细介绍。

返回

jqXHR对象。

jQuery 3:新增签名

jQuery 3为\$.get()工具函数新增了一个新的签名:

\$.get([options])

options可以是拥有多个属性的对象。要获取更多的信息,可以参考本章后面面对\$.ajax()的详细介绍。值得注意的是method属性,options对象会自动设置为"GET"。

\$.get()工具函数允许我们以更多的方式执行GET请求。除了请求参数、成功应答后的回调函数,还可以设置应答消息处理的格式,以及传递给回调函数的格式。如果方式不够多,还可以使用\$.ajax()工具函数,dataType参数的控制更加细致。现在,根据应答消息的内容类型,默认它为html或xml。通过在靴子商店网页(Boot Closet page)使用\$.get(),可以取代之前的load()方法,新的代码如列表10.5所示。

列表 10.5 使用 GET 方式修改靴子商店网页(Boot Closet page)代码

```
$('#boot-chooser-control')
    .change(function(event) {
        $.get(
            'actions/fetch-product-details.php',
            {
                model: $(event.target).val()
            },
            function(response) {
                注入结果 HTML ➡ $('#product-detail-pane').html(response);
                $('#[value=""]', event.target).remove();
            }
        );
    });
```

← ① 执行 GET 请求

页面中的改动很小,但是非常重要。使用\$.get()①取代load(),传递的是相同的URL,以及相同的请求参数。因为这个例子中使用了\$.get()函数,所以即使传递了一个对象,也可以

确保GET请求被执行。`$.get()`捕获自动把应答结果注入DOM元素中，所以需要通过jQuery的`html()`方法来实现②。

新版本的例子代码可以在文件[http://localhost\[:8080\]/chapter-10/phase.2.html](http://localhost[:8080]/chapter-10/phase.2.html)中找到。在浏览器中打开页面，选择靴子演示，可以看到发送了GET请求，如图10.5所示。

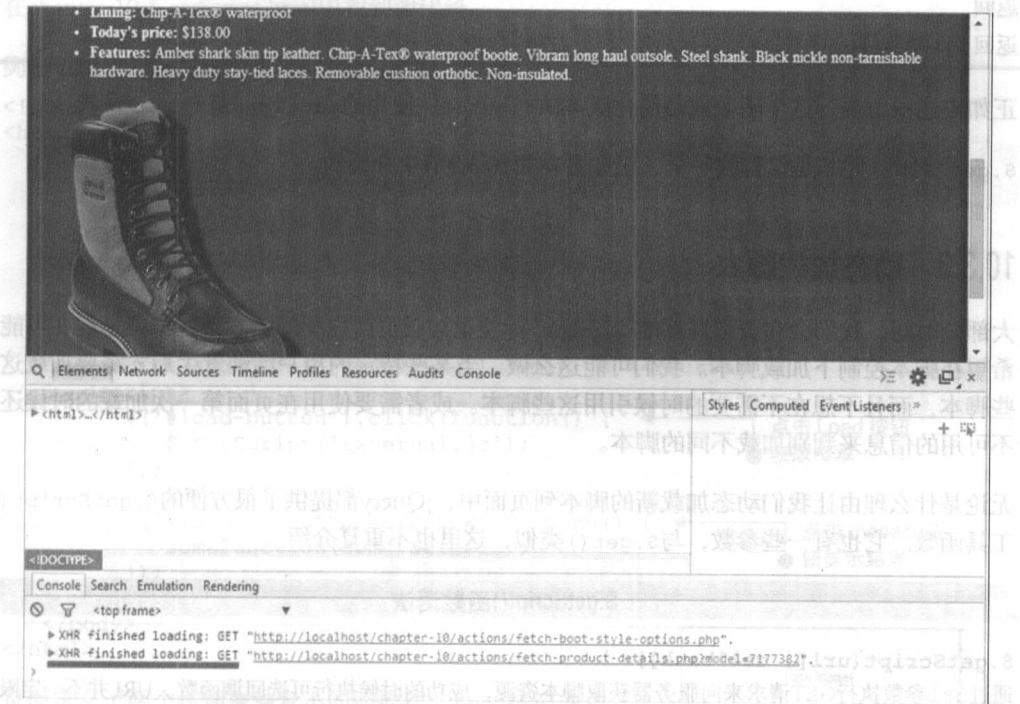


图 10.5 作为适当的操作，现在可以看到第二个请求是 GET 请求而不是 POST 请求

在这个例子中，从服务端返回的是HTML代码，并将其插入DOM元素中，正如你看到的，`$.get()`的`dataType`参数的可用值，除了HTML，还有很多其他格式。

现在来看另外一个有用的jQuery工具，它在需要使用JSON格式数据的时候非常有用。

10.3.2 获取 JSON 数据

XML太过繁重，又没有其他合适的数据传输格式，JSON常用来传输数据。其中一个原因就是JSON易于在客户端脚本里处理，而且jQuery简化了处理工作。

当知道返回结果是JSON时，`$.getJSON()`工具函数会自动解析返回的JSON字符串，然后传递给回调函数。`$.getJSON()`函数的语法如下，其参数与`$.get()`函数的一样，这里不再重复介绍。

\$.getJSON()函数语法

\$.getJSON(url[, data][, callback])

使用指定的URL向服务器发送请求，附加需要的参数作为查询字符串。应答消息是JSON格式的字符串，结果会传递给回调函数。

返回

返回jqXHR实例。

正如描述中所说，这个函数就是简化版本的\$.get()，使用的dataType是"json"格式。

\$.getJSON()并非是仅有的一个，正如会在第10.3.3节中介绍的。

10.3.3 动态加载脚本

大部分时间，我们会在页面底部的<script>标签里加载页面需要的脚本文件。但是现在可能希望在脚本控制下加载脚本。我们可能这么做，当某些特定的用户活动发生后才需要加载这些脚本，而且不想在不需要的时候引用这些脚本。或者需要使用在页面第一次加载的时候还不可用的信息来判别加载不同的脚本。

无论是什么理由让我们动态加载新的脚本到页面中，jQuery都提供了很方便的\$.getScript()工具函数。它也有一些参数，与\$.get()类似，这里也不重复介绍。

\$.getScript()函数语法

\$.getScript(url[, callback])

通过url参数执行GET请求来向服务器获取脚本资源，成功的时候执行可选回调函数。URL并不一定限制于当前域名。

返回

返回jqXHR实例。

在底层，此函数通过设置\$.get()的data参数为undefined，或者设置dataType参数为"script"。\$.getScript()的函数代码定义如下：

```
getScript: function( url, callback ) {  
    return jQuery.get( url, undefined, callback, "script" );  
}
```

当执行此函数时，文件中的脚本会被评估，每行脚本都会被执行，而且任意定义的变量或者函数都变得可用。

实际看下例子代码。思考下面的脚本文件（chapter-10/external.js文件）：

```
alert('I am inline!');  
var someVariable = 'Value of someVariable';
```

```
function someFunction(value) {
    alert(value);
}
```

这个小脚本文件保护了一些语句（当执行代码的时候弹出警告框）、一个变量声明，以及一个弹出警告框的函数声明。现在编写代码让页面动态加载脚本。页面如列表10.6所示，可以在chapter-10/\$.getScript.html中找到源代码。

列表 10.6 动态加载脚本文件并检查结果

```
<!DOCTYPE html>
<html>
  <head>
    <title>$.getScript() Example</title>
    <link rel="stylesheet" href="../../css/main.css"/>
  </head>
  <body>
    <button id="load-button">Load</button>
    <button id="inspect-button">Inspect</button>
    <script src="../../js/jquery-1.11.3.min.js"></script>
    <script>
      $('#load-button').click(function() {
        $.getScript('external.js');
      });

      $('#inspect-button').click(function() {
        someFunction(someVariable);
      });
    </script>
  </body>
</html>
```

页面定义了两个按钮①触发不同的操作。Load按钮使用\$.getScript()动态加载外部的“external.js”文件②。点击这个按钮，内联函数语句会执行，并弹出警告框，如图10.6所示。

点击“Inspect”按钮会执行处理器③，当执行动态加载的函数someFunction()时，传递的是动态加载的someVariable变量。如果警告框如图10.7所示，那么变量和函数都正确加载了。



图 10.6 动态加载并评估脚本文件，执行代码

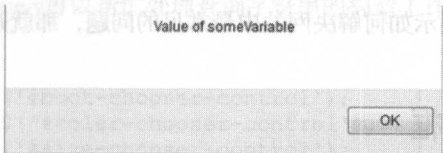


图 10.7 出现警告框表示动态函数加载正确，且正确显示了动态加载的变量的值

除了提供执行GET请求的工具函数以外，jQuery还提供了发送POST请求的函数。下面看看具体内容。

10.3.4 发送 POST 请求

选择POST而不是GET的原因有很多。首先，HTTP规范定义的POST用于处理非幂等请求。因此，如果请求可能导致服务端状态变化，导致应答消息变化，就应该使用POST。此外，除了实践原则和约定以外，POST操作还用来处理向服务器发送大量数据的情况，数据超过了URL查询字符串的限制——浏览器查询字符串的长度限制。而且有时候服务器端资源需要根据使用GET或者POST方法提交的请求来执行不同的功能。这些只是我们要选择POST而不是GET请求的原因。

对于需要使用POST的场景，jQuery提供了`$.post()`工具函数。这个函数与`$.get()`类似，不同的是使用了HTTP POST方法。为此，在方法的介绍部分，我们不会再列举重复的部分。`$.post()`函数的语法介绍如下。

`$.post()`函数语法

`$.post(url[, data][, callback][, dataType])`

使用指定的URL地址向服务器发送POST请求，附带任意参数，在请求消息体中传递。

返回

jqXHR实例。

jQuery 3: 新增签名

jQuery 3为`$.post()`工具函数新增了签名：

`$.post([options])`

`options`是一个拥有多个属性的对象。更多信息请参考本章后面有关`$.ajax()`的详细信息。

值得注意的是，`options`对象的`method`属性会自动设置为“POST”。

jQuery会处理传递的请求数据，把它们封装到请求消息体中（与查询字符串相对应），并设置正确的HTTP方法。

现在，再回到靴子商店（Boot Closet）项目，你已经有了很好的开始，但是还有一项购买靴子的工作要做，不仅仅要选择样式，顾客还想要选择自己喜欢的颜色，而且需要指定尺寸。我们会使用额外的需求来展示如何解决网站中最常见的问题，那就是……

10.3.5 实现级联下拉框

实现级联下拉框——后面的下拉框选项取决于前一个下拉框的值——这是Web网站中最常用的模式。本节会为靴子商店（Boot Closet）实现这个功能，证明jQuery实现这种功能有多简单。

我们已经学习了使用动态服务端数据加载下拉框数据，现在要尝试把多个下拉框放到一起，

并且强调它们之间的级联关系，这里仅多做了点工作。

下面列举下一阶段需要完成的页面开发工作。

- 添加颜色和尺寸的下拉框。
- 当选择样式时，添加颜色下拉框，并显示该样式可用的颜色。
- 当选择颜色时，添加尺寸下拉框，并显示该样式和颜色下可用的尺寸。
- 确保数据的一致性。这里包括当选择的时候，从新创建的下拉框里删除“- please make a selection -”选项，而且确保三个下拉框不会显示无效的组合。

再继续使用load()方法，这次强迫它使用GET而不是POST方式。原因是当使用Ajax加载HTML代码时，load()方法看起来更自然。

开始的时候，要看看为其他下拉框定义的HTML标签代码。

选择元素的新容器内定义了三个标签元素：

```
<div id="selections-pane">
  <label for="boot-chooser-control">Boot style:</label>
  <select id="boot-chooser-control" name="model"></select>
  <label for="color-chooser-control">Color:</label>
  <select id="color-chooser-control" name="color" disabled></select>
  <label for="size-chooser-control">Size:</label>
  <select id="size-chooser-control" name="size" disabled></select>
</div>
```

颜色的下拉菜单，
初始化为禁用

模型的下
拉菜单

尺寸的下拉菜单，
初始化为禁用

284

保留了之前的select元素，但是新增了两个下拉框元素：一个是颜色，一个是尺寸，每个元素都初始化为空、禁用状态（使用disabled属性）。

样式选择下拉框要执行两个职责。不仅包括选择后获取和显示靴子的详细信息，还包括生成可用的颜色选项供用户选择。

先来重构一些获取信息的代码。如果想使用load()方法，但是也想强制使用GET，那么与最早初始化的POST相反。为了让load()支持GET方式，需要传递字符串而不是对象来传输请求参数数据。幸运的是，jQuery可以帮忙处理查询字符串的构造工作。样式下拉框change事件处理器的第一步重构如下：

```
var $bootChooser = $('#boot-chooser-control');
var $colorChooser = $('#color-chooser-control');
var $sizeChooser = $('#size-chooser-control');
```

全部代码使用的变量

```
$bootChooser.change(function() {
  $('#product-detail-pane').load(
    'actions/fetch-product-details.php',
```

```

        $(this).serialize() ← 以查询字符串方式提供数据
    );
    // 更多的代码
});

```

使用jQuery的serialize()方法,可以创建一个表示样式下拉框值的字符串,因此,强制load()使用GET方式获取数据。

事件处理器的第二个职责就是要为颜色选择下拉框添加适当的值,并启用它。看看为事件处理器添加的剩下部分的值:

```

$colorChooser.load(
    'actions/fetch-color-options.php',
    $(this).serialize(),
    function() {
        $(this).prop('disabled', false); ← ② 启用颜色控件
        $sizeChooser
            .prop('disabled', true) ← ③ 启用并清空尺寸控件
            .html('');
    }
);

```

这些代码看起来很熟悉。这就是load()方法的另外一种使用方式,这时可从actions/fetch-color-options.php页面加载数据,它的作用就是返回可用颜色值的下拉选项集合①。

还需要指定回调函数,当GET请求成功返回数据后执行。在回调函数内,执行两个重要的任务。第一,启用颜色选择空间②。调用load()注入<option>标签,一旦填充,如果不启用,那么仍然是禁用状态。第二,回调函数禁用并清空尺寸选择控件③。为什么?(暂停一会想一想。)

虽然尺寸控件在第一次样式和颜色改变时已经被禁用并清空,但是后面呢?如果用户选择样式和颜色之后(会看到尺寸控件的填充结果),用户改变选择的样式呢?尺寸是根据样式和颜色的组合来显示的,所以之前显示的尺寸不一定与选择的新样式和颜色一致。因此,无论何时修改样式,都需要清空尺寸选项,重置尺寸控件为初始条件状态。

在想要松口气放松之前,还有一些工作要做。仍然要告诉颜色选择下拉框使用选择的样式和颜色值来获取对应的尺寸数据。代码处理方式类似,如下:

```

$colorChooser.change(function() {
    $sizeChooser.load(
        'actions/fetch-size-options.php',
        $colorChooser
            .add($bootChooser)
            .serialize(),
        function() {
            $(this).prop('disabled', false);
        }
    );
});

```

在颜色控件的change事件里,可以通过actions/fetch-size-options.php获取对应的尺寸

信息，以传递靴子的样式和颜色值，并且启用尺寸控件。

还有一件事情需要我们来做。当每个下拉框没有初始化的时候，它会被空值的option元素清空，并且显示“- choose a something -”（选择一个）信息。还记得这个页面的前一个阶段，根据选择从样式下拉框里删除这个选项。

好了，可以为样式和颜色下拉框的change事件处理器添加一样的代码，以及尺寸下拉框（当前还没有）。我们尽量完善一些功能。

许多Web开发者经常忽略的一个事件模型的功能就是事件冒泡（event bubbling）。页面开发者通常只关注在事件目标上，而忘记事件会沿着DOM树冒泡传播，事件处理器可以在目标元素层之外的地方处理它。

286

如果意识到从三个下拉框里删除空白值的选项都可以使用相同的方式处理，而不需要关注事件目标元素，就可以避免在三个地方编写重复的事件处理器代码了，更高层的DOM元素已处理这个事件。这个知识点应该会让我们想起第6章里介绍的事件委托机制。

回顾文档的结构，三个下拉框包含在ID为selections-pane的<div>元素里。可以使用下面单个事件处理器来处理三个下拉框临时选项的删除工作：

```
$('#selections-pane').change(function(event){
    $('[value=""]', event.target).remove();
});
```

当任意一个下拉框的change事件发生改变时，这个侦听器会被触发，然后目标事件元素会删除空值的option选项（改变的下拉框）。

有了这些代码，就已经完成了靴子商店（Boot Closet）第三阶段的代码，添加了级联下拉框，如图10.8所示。在任意新下拉框需要依赖前一个下拉框值的页面中，都可以使用相同的方法。这个阶段的例子代码可以在[http://localhost\[:8080\]/chapter-10/phase.3.html](http://localhost[:8080]/chapter-10/phase.3.html)中找到。

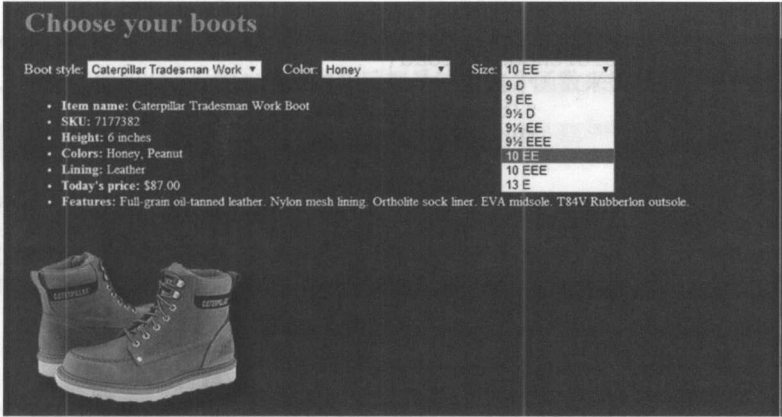


图 10.8 靴子商店（Boot Closet）的第三阶段，展示了实现级联下拉框多么简单

287

页面的完整代码如列表10.7所示。

列表 10.7 靴子商店 (Boot Closet) 现在支持级联下拉框

```
<!DOCTYPE html>
<html>
  <head>
    <title>The Boot Closet - Phase 3</title>
    <link rel="stylesheet" href="../../../css/main.css" />
    <link rel="stylesheet" href="../../../css/bootcloset.css" />
  </head>
  <body>
    <div id="banner"></div>
    <h1>Choose your boots</h1>
    <div>
      <div id="selections-pane">
        <label for="boot-chooser-control">Boot style:</label>
        <select id="boot-chooser-control" name="model"></select>
        <label for="color-chooser-control">Color:</label>
        <select id="color-chooser-control" name="color" disabled></select>
        <label for="size-chooser-control">Size:</label>
        <select id="size-chooser-control" name="size" disabled></select>
      </div>
      <div id="product-detail-pane"></div>
    </div>

    <script src="../../../js/jquery-1.11.3.min.js"></script>
    <script>
      var $bootChooser = $('#boot-chooser-control');
      var $colorChooser = $('#color-chooser-control');
      var $sizeChooser = $('#size-chooser-control');

      $bootChooser
        .load('actions/fetch-boot-style-options.php')
        .change(function() {
          $('#product-detail-pane').load(
            'actions/fetch-product-details.php',
            $(this).serialize()
          );

          $colorChooser.load(
            'actions/fetch-color-options.php',
            $(this).serialize(),
            function() {
              $(this).prop('disabled', false);
              $sizeChooser
                .prop('disabled', true)
                .html('');
            }
          );
        });

      $colorChooser.change(function() {
        $sizeChooser.load(
          'actions/fetch-size-options.php',
          $colorChooser
            .add($bootChooser)
```

```

        .serialize(),
        function() {
            $(this).prop('disabled', false);
        }
    );
});

$('#selections-pane').change(function(event) {
    $('[value=""]', event.target).remove();
});
</script>
</body>
</html>

```

正如我们看到的，使用load()方法和各种不同的GET和POST jQuery Ajax函数，可以对请求进行更复杂的控制。但是对于要完成控制Ajax请求的情况，jQuery也提供了想要的方法。

10.4 完全控制 Ajax 请求

Taking full control of an Ajax request

目前为止介绍的函数和方法可以满足大多数情况的需求，但有时候希望自己完全控制请求。例如，想确保每次都执行Ajax请求、收到新数据（避免浏览器缓存）。

另外一种情况，低级别方法可能派上用场，就是当用户执行Ajax请求但是结果必须在特定的时间范围内获取到。要提的最后一个例子，有时需要Ajax调用结果是特定的格式，例如，纯文本，但是想把它转换为其他格式，比如HTML或者XML。

本节将会学习jQuery如何帮助我们处理这种问题。

10.4.1 使用全部参数发送 Ajax 请求

有时想对Ajax进行更细力度的控制，jQuery提供了通用的工具函数来发送Ajax请求：\$.ajax()。从底层看，所有其他的jQuery的Ajax请求背后都是使用这个函数来执行的。它的语法如下。

\$.ajax()函数语法

\$.ajax(url[, options])

\$.ajax([options])

使用URL和参数选项执行Ajax请求，控制请求发送和回调提醒。在此函数的第二版里，URL也在参数选项里指定。如果没有参数，请求就发送给当前页面。

参数

url(String) 请求发送的URL地址。

options(Object) 一个对象，它的属性定义此Ajax请求的参数。

参考表10.2的详细信息。

返回

jqXHR实例。

看起来很简单，不是吗？但是，不要被欺骗了。options参数可以指定许多参数来控制函数的操作，包括发送请求的URL。这些选项（按重要性和使用性排序）定义如表10.2所示。

表 10.2 \$.ajax() 工具函数的参数选项

名 字	描 述
url	(String)字符串包含请求要发送的 URL 地址。如果为空字符串，请求就发送给当前的 URL。
method	(String)使用的 HTTP 动词，通常是 POST 和 GET。如果忽略，则默认使用 GET。如果使用 jQuery 1.9.0 之前的版本，则属性必须使用 type 参数来单独设置。
data	(String Object Array)定义要发送给服务器的数据。如果是 GET 方式，则值会转换为查询字符串。如果是 POST，则值会作为请求消息体传输。\$.ajax() 工具函数会处理编码工作。 参数可以是字符串，它被用来当做查询字符串、应答消息体、属性给序列化的对象或者包含 name 和 value 的数组对象。
dataType	(String)在基本形式上，它是用来表示返回数据类型的关键字。它的值决定了返回给回调函数的数据类型。有效值如下： xml: 返回的应答文本作为 XML 文档进行解析，结果 XML DOM 会传递给回调函数。 html: 返回的应答文本直接传递给回调函数，里面包含的任何<script>代码块都会被评估执行。 json: 返回的应答文本作为 JSON 字符串解析，对象传递给回调函数。 jsonp: 与 json 类似，除了远程脚本外，假设远程服务器支持它。 script: 应答文本传递给回调函数。在回调函数调用之前，应答消息作为 JavaScript 语句处理。 text: 应答文本作为纯文本处理。 服务器负责设置 content-type 应答消息头。默认值通过 jQuery 从应答消息获取的信息来确定，为 xml、json、script 或 html 之一。 值也可以是空格分割的字符串。此时，jQuery 会把数据类型转换为其他类型。例如，如果应答消息是文本，希望作为 XML 处理，则可以编写为"text xml"。
cache	(Boolean)如果是 false，则浏览器不会缓存应答消息。注意，它只对 HEAD 和 GET 请求有效。除了 dataType 设置为 script 或 jsonp，默认是 true。
context	(Object Element)指定一个对象或者 DOM 元素作为回调函数的上下文。默认上下文是一个表示 Ajax 请求设置的对象。
timeout	(Number)设置 Ajax 请求超时的毫秒值。从\$.ajax() 调用开始计时。如果请求在超时前没有完成，则请求会终止，并且触发错误回调函数（定义的话）。

名 字	描 述
global	(Boolean)如果是 false, 则禁用触发全局的 Ajax 事件。这就是 jQuery 在处理 Ajax 请求时触发自定义事件的不同点和条件, 我们会在后面详细讨论。如果忽略, 则默认(true)启用触发全局事件
contentType	(String)指定请求的内容类型 (content type)。如果忽略, 则默认是 application/x-www-form-urlencoded; charset=UTF-8, 此类型也可作为默认提交表单的内容类型
success	(Function Array)当请求成功时调用的一个函数或者函数数组。应答消息体返回作为函数的第一个参数, 并且根据 dataType 属性值进行评估。第二个参数是包含状态值的字符串——这种情况下, 一直是 "success"。第三个参数提供了 jqXHR 实例的引用
error	(Function Array)当请求失败时调用的一个函数或者函数数组。传递三个参数给函数: jqXHR 实例、状态消息 (此时是 "error"、"timeout"、"abort" 或 "parseerror"), 以及一个可选的异常对象, 有时候从 jqXHR 对象返回。对于跨域脚本和跨域 JSONP 请求, 不会调用错误处理器函数
complete	(Function Array)当请求完成时调用的一个函数或者函数数组。传递两个参数给函数: jqXHR 实例和状态消息之一 ("success"、"error"、"notmodified"、"timeout"、"abort" 和 "parseerror")。如果指定了 success 或者 error 回调函数, 则此函数会在它们后面执行
beforeSend	(Function)函数在初始化请求之前调用。这个函数传递了 jqXHR 实例, 可以用来设置自定义头或者指定其他的预定义操作。返回 false 会取消 Ajax 请求
async	(Boolean)如果为 false, 则请求会同步提交。默认情况下值是 true, 请求是异步模式。跨域请求和 dataType: "jsonp" 请求不支持异步操作
processData	(Boolean)如果设置为 false, 则阻止数据被处理为 URL 编码格式。默认情况下值是 true, data 的值是 URLencoded, URL 编码为合适的格式给 application/x-www-form-urlencoded 请求使用
contents	(Object)基于给定的内容类型, 字符串/正则表达式对用来决定 jQuery 如何解析应答消息
converters	(Object)一个包含 dataType-to-dataType 转换器的对象。每个转换器的值是一个可以返回转换结果的函数
crossDomain	(Boolean)设置为 true, 会强制向同域名发送 crossDomain 跨域请求。默认同域名是 false, 跨域名请求是 true
headers	(Object)与请求一起发送的额外的键/值对, 默认为空对象
dataFilter	(Function)过滤应答消息的回调函数过滤器。此函数接受原始的应答数据和 dataType 参数, 返回过滤后的数据
ifModified	(Boolean)如果为 true, 根据 Last-Modified 消息头, 只有当上一个请求应答消息内容没有改变时, 才允许请求继续。如果忽略, 则不会执行头部检查。默认为 false
isLocal	(Boolean)允许当前环境为本地(例如 filesystem)。jQuery 默认识别作为 local 的协议是 file、*-extension 和 widget

名 字	描 述
jsonp	(String)指定查询参数名来重写 callback 的默认的 jsonp 回调参数
jsonpCallback	(String Function)指定 JSONP 请求的回调函数名。这个值用来替换 jQuery 自动生成的随机名称
username	(String)在 HTTP 验证请求中使用的用户名
password	(String)在 HTTP 验证请求中使用的密码
scriptCharset	(String)当远程和本地内容字符集不同时,用来设置 script 和 jsonp 请求的字符集
statusCode	(Object)包含 HTTP 状态码数值的对象,函数可以根据不同的状态码进行处理。默认为空
xhr	(Function)提供对 XHR 实例自定义实现的回调函数
xhrFields	(Object)设置到原生 XHR 对象的名/值对。默认对象为空
accepts	(Object)请求头里发送给服务器的内容类型,告诉服务器客户端接收什么内容。默认情况下,取决于 dataType 的值
mineType	(String)用来重写 XHR mime 类型的 mime 类型
traditional	(Boolean)如果为 true,则使用传统风格序列化参数。参考第 9 章 \$.param() 的关于参数序列化介绍的内容

不要被这个列表吓到。我们知道内容可能有点多,但是首先,并不需要记住这些参数选项(本书和官方文档可以作为参考手册);其次,很少有请求会用到全部参数。

JSONP 是关于什么的?

JSON 是轻量级且广泛使用的数据交换格式。网站通常使用 XHR 对象和此格式来执行 Ajax 请求。这种机制遵循同源策略,这意味着只有请求页面和数据源是同一个域名才会执行请求。要突破这个限制,一个新的名为 JSONP 的机制发明出来,2005 年, Bob Ippolito 在他的文章《Remote JSON-JSONP》(<http://bob.ippoli.to/archives/2005/12/05/remote-json-jsonp/>)中提出了这个概念。

JSONP (“JSON with padding”的简称)通过创建 script 元素工作(在 HTML 标签中或者通过 JavaScript 插入 DOM 中)。通过引用返回 JSON 数据的外部资源、页面中声明的函数来执行请求,名字通过 script 元素设置。通常函数名使用 callback 参数名设置。例如,可以创建 script 元素,代码如下:

```
<script src="http://www.example.com/data?callback=myFunction"></script>
```

这个例子中, myFunction() 函数是一个定义在页面中的函数,它执行请求处理并返回 JSON 数据。服务器可以处理这种请求,应答消息如下:

```
myFunction({"name": "jQuery in Action"});
```

这会导致页面使用服务器返回的数据作为参数来执行`myFunction()`函数。

更多关于JSONP的内容，请访问网站www.json-p.org。

“没有使用`$.ajax()`的例子？”我们听到了你的疑问，不要担心；第11章会专门讲解Ajax网站项目开发。

有时候，如果可以对表10.2中列举的参数选项设置默认值，那该多好，尤其是要发送大量请求的页面。下面看看如何实现。

293

10.4.2 设置默认请求

jQuery提供了设置Ajax默认请求属性的方法，在不需要重写默认值的时候使用。这会让页面初始化相似的Ajax请求更加容易。这个设置Ajax默认参数列表的函数就是`$.ajaxSetup()`。

`$.ajaxSetup()`函数语法

`$.ajaxSetup(options)`

设置传递的参数选项值作为默认的`$.ajax()`调用请求的参数，或者继承函数`$.get()`和`$.post()`，以及第三方库或者插件执行Ajax函数的默认参数。

参数

`options(Object)` 对象定义了Ajax默认参数选项的值。这些描述为同一属性，如表10.2所述。

这个函数不能用来为`success`、`error`、`completion`设置回调处理器（很快会看到另外一种设置方法）。

返回

`undefined`。

在脚本处理的任何一点，这个函数都可以用来设置`$.ajax()`调用的默认参数。此方法必须谨慎使用，因为它会改变页面上其他`$.ajax()`执行的Ajax请求或者类似方法的默认参数。

注意：使用此方法不会修改`load()`的默认设置。而且对于`$.get()`和`$.post()`工具函数，HTTP方法不能通过这些默认值重新设置，例如，使用HTTP GET方法。

假设现在开发一个页面，对于大部分的Ajax请求（是使用工具函数而不是`load()`方法），我们希望设置一些默认参数，而不需要为每个调用都重新指定。可以在`script`元素里编写下面的第一行代码：

```
$.ajaxSetup({
  type: 'POST',
  timeout: 5000,
  dataType: 'html'
});
```

294 这会确保后续的Ajax请求（除了之前强调的）使用这些默认值，除非显示给调用Ajax请求的函数传递新的参数值。尤其是让所有的请求都使用POST方式，请求的超时设置为5秒，应答消息作为HTML格式进行处理。

现在来看看之前提到的控制全局事件的global参数选项。

10.4.3 处理 Ajax 事件

在Ajax请求执行过程中，jQuery触发了一系列事件，可以根据需要建立事件处理器来通知请求进程或者执行不同的操作。jQuery把这些事件分为局部事件和全局事件。

局部事件(local events)可以通过直接为\$.ajax()的beforeSend、success、error、complete选项指定回调函数，或者间接提供回调方法（通过轮流使用\$.ajax()函数执行请求）来实现。我们可能一直在处理局部事件，但是不知道它的存在，已经为jQuery Ajax函数注册了回调函数。

全局事件(global events)被页面上任意的Ajax请求触发。可以通过document的on()方法（其他元素无法工作）建立事件处理器。全局事件和局部事件类似，对应关系是ajaxStart、ajaxSend、ajaxSuccess、ajaxError、ajaxStop和ajaxComplete。

附加的事件处理器接收三个参数：jQuery.Event实例、jqXHR实例、参数选项对象。表10.3按照事件顺序列举了这些事件。

表 10.3 jQuery 事件类型

事件名称	类型	描述
ajaxStart	Global	在 Ajax 请求开始时触发事件，只要没有其他活动的请求。对于并发请求，这个事件只为第一个请求触发。只传递 jQuery.Event 实例
beforeSend	Local	在发送请求之前调用，允许修改 XHR 实例，可以通过返回 false 来取消请求
ajaxSend	Global	在发送请求之前触发，允许修改 XHR 实例
success	Local	当请求返回成功应答消息时调用
ajaxSuccess	Global	当请求返回成功应答消息时触发
error	Local	当请求返回错误应答消息时调用
ajaxError	Global	当请求返回错误应答消息时触发。可选的第四个参数引用了发生的错误对象
complete	Local	当请求完成时调用，无论状态结果是什么。这个回调函数也会调用同步请求
ajaxComplete	Global	当请求完成后触发，无论状态码是什么。回调函数也会调用同步请求
ajaxStop	Global	当 Ajax 请求完成并且没有其他并发活动的请求时触发。只传递 jQuery.Event 实例

为了确保概念清楚，我们想强调的是，局部事件表示传递给\$.ajax()的回调函数；而全局事件是触发的自定义事件，且可以通过建立处理器来处理（为document对象），和其他事件类型一样。

表10.3介绍了ajaxStart和ajaxStop，它们接收jQuery.Event作为唯一参数。这个参数没有真实的使用场景，所有官方文档里没有说明。但是，为了确保精确，要详细介绍下（因为下面的demo里要使用）。也可以在<https://github.com/jquery/api.jquery.com/issues/478>中阅读更多资料。

除了使用on()来建立事件处理器，jQuery还提供了许多其他方法来建立处理器，语法如下。

jQuery Ajax event establishers方法语法

```
ajaxComplete(callback)
ajaxError(callback)
ajaxSend(callback)
ajaxStart(callback)
ajaxStop(callback)
ajaxSuccess(callback)
```

通过方法名为jQuery Ajax事件指定传递的回调函数作为事件处理器。

参数

callback(Function) 此函数可以作为Ajax事件处理器。函数上下文(this)是建立事件处理器的DOM元素。参数可通过表10.3列出。

返回

jQuery集合。

现在把这些方法统一在一个简单的例子里使用，看看如何轻易实现跟踪Ajax请求进度。测试页面的布局如图10.9所示，可以在[http://localhost\[:8080\]/chapter-10/ajax.events.html](http://localhost[:8080]/chapter-10/ajax.events.html)中找到例子文件。

页面有三个控件：一个count字段、一个Good request（好请求）按钮和一个Bad request（坏请求）按钮。这些按钮会按照设置的数量发送请求。Good request按钮请求有效的资源，而Bad request按钮请求无效的资源，这些请求最终会失败。



图 10.9 用来检查 jQuery Ajax 事件的页面初始状态，通过激发多个事件并观察处理器实现

在页面代码里，可以定义如下多个事件处理器：

```

var $log = $('#log');
$(document).on(
    'ajaxStartajaxStopajaxSendajaxSuccessajaxErrorajaxComplete',
    function(event) {
        $log.text($log.text() + event.type + '\n');
    }
);

```

这段代码在document对象上为每个不同的jQuery Ajax事件类型建立了处理器。事件处理器在ID为log的textarea元素里写入事件类型信息。

设置请求数量为1, 点击“Good request”按钮, 观察结果。我们会看到每个jQuery Ajax事件会按照表10.3中的顺序来触发。但是要通过理解ajaxStart和ajaxStop的行为来设置请求数量为2, 点击“Good request”按钮, 会看到如图10.10所示的结果。

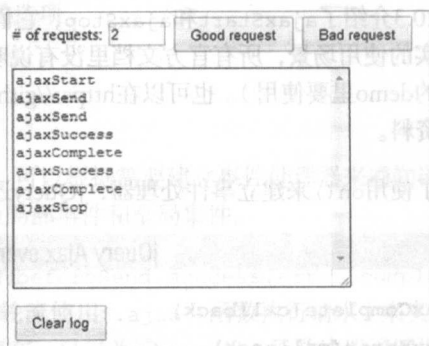


图 10.10 当有多个活动请求时, ajaxStart 和 ajaxStop 事件为所有请求只执行一次

这里可以看到过程, 当发送多个请求时, ajaxStart和ajaxStop事件为所有的并发请求只触发一次, 而其他类型的事件会每个请求触发一次。

297

点击“Bad request”按钮, 观察事件行为, 会获得如图10.11所示的结果, ajaxError事件被触发。

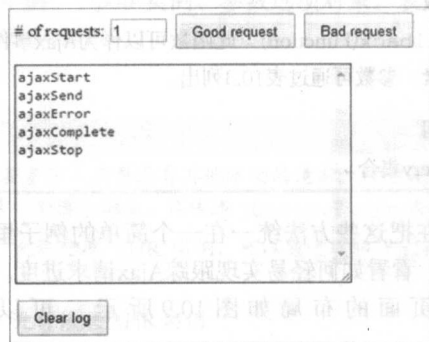


图 10.11 坏请求的结果展示了 ajaxError 事件被触发

正如你所看到的, \$.ajax() 给了我们许多参数选项, 提供了更大的灵活性, 但是某些时候希望有更多的控制。例如, 也许想要根据某些选项处理请求, 或者在发送请求之前修改现有的值, 或者管理Ajax请求传输的数据。一种可能的使用情况, 正如将在第10.4.4节例子里讨论的, 阻止Ajax请求访问禁止的域名。下面看看jQuery提供了哪些机制。

10.4.4 高级 Ajax 工具函数

除了目前为止我们讨论的所有方法和工具函数, jQuery还提供了其他的工具函数。可能大家都没有太多机会看到这两个函数, 但是因为它们的存在, 这里就介绍给大家。向\$.ajaxPrefilter()和\$.ajaxTransport()打招呼(你好)吧。

当调用\$.ajax()时, \$.ajaxPrefilter()会根据某些选项参数阻止Ajax请求, 语法如下。

\$.ajaxPrefilter([dataTypes,] callback) (callback) 在请求发送之前, 或者在调用\$.ajax()之前, 处理自定义Ajax选项参数, 或者修改已有选项。

参数

dataTypes(String) 可选字符串, 包含一个或者多个空格分隔符dataTypes, 和表10.2中\$.ajax()的描述一样。如果传递参数, 只有在匹配请求参数dataTypes时调用处理器。

callback(Function) 未来Ajax请求默认的回调函数。此函数接收三个参数: options, 包含最近的请求参数选项; originalOptions, 存储提供给\$.ajax()的参数选项(没有从ajaxSettings中默认); jqXHR, 请求的jqXHR对象。

返回

未定义。

298

要看下具体的例子代码, 假设想要终止到某个域名的所有XML类型请求。也许我们想要做的原因是知道这些请求会一直失败。

要实现这个目标, 可以编写如下代码, 代码在 [http://localhost:8080/chapter-10/\\$.ajaxPrefilter.html](http://localhost:8080/chapter-10/$.ajaxPrefilter.html)和JS Bin(<http://jsbin.com/bitiv/edit?js,console>)中可以找到:

```
$.ajaxPrefilter('xml', function(options, originalOptions, jqXHR) {  
    if ($.inArray(options.url, originalOptions.deniedDomains) !== -1) {  
        console.log('Ajax request to ' + options.url + ' aborted');  
        jqXHR.abort();  
    } else {  
        console.log('Ajax request performed');  
    }  
});  
  
$.ajax(  
    'http://www.google.com',  
    {  
        dataType: 'xml',  
        deniedDomains: [  
            'http://www.google.com',  
            'http://www.manning.com'  
        ]  
    }  
);
```

如果域不允许, 则终止请求

执行 Ajax 请求

根据 dataType 和禁用域名过滤请求

这个函数的目标并不局限于根据选项参数来修改Ajax请求, 它也可以用在想要从最初的dataType重定向到另外一个参数的时候, 通过返回想要的dataType来实现。

要介绍的另外一个生僻的函数就是\$.ajaxTransport()。它是一个低级别的函数, 允许我们控制\$.ajax()如何传输请求的数据。其语法如下。

\$.ajaxTransport([dataType,] callback)
创建对象来处理实际的Ajax数据传输。

参数

dataType (String) 可选字符串参数, 包含使用的数据类型。如果传递了参数, 处理器只有在匹配请求的dataType时才调用。

callback(Function) 一个函数, 使用dataType来返回新的传输对象。此函数接收三个参数: options, 包含请求选项; originalOptions, 存储提供给\$.ajax()调用参数选项(默认不会从ajaxSettings获取默认值); jqXHR, 请求的jqXHR对象。

返回

299 未定义。

callback回调函数必须返回一个新的传输对象, 它是JavaScript对象, 提供两个方法, 即send()和abort(), 其内部都使用了\$.ajax()。

send()函数接收两个参数, 即headers和completeCallback。前者是一个包含键/值对的对象, 包含在请求消息头里; 而后者是当请求完成的时候用来通知\$.ajax()的回调函数。

使用最后一个复杂的工具函数, 我们已经学习了jQuery提供处理Ajax请求的方法和函数。展示所有的例子是一个好的开始, 都是为了帮助理解jQuery如何处理Ajax请求。但是你, 亲爱的读者, 应该配得上更多的知识。

第11章的目标就是展示一个真实的项目例子, 它使用了强大的Ajax来解决可能遇到的公共问题——也许已经遇到过了。

10.5 总结

Summary

Ajax技术是现代网站的关键部分, 而jQuery给我们提供了丰富的工具。

要加载HTML内容到DOM元素里, load()方法可以提供从服务器获取内容的简单方式, 并可以轻易赋值给匹配的元素。是使用GET或者POST, 可以根据提供给data参数的类型来确定。

当要使用GET时, jQuery提供了\$.get()和\$.getJSON()工具函数。当要从服务器返回JSON数据时, 可以使用\$.getJSON()。要强制使用POST方式, 可以使用\$.post()工具函数。

如果需要最大的灵活性, 则可以使用\$.ajax()工具函数, 该函数带有丰富的参数, 允许我们控制Ajax请求的几乎每个细节。jQuery里所有的Ajax功能背后都使用了这个函数来提供其功能。

为了减少管理参数选项设置带来的麻烦，jQuery提供了\$.ajaxSetup()工具函数，允许我们为经常使用的\$.ajax()函数配置默认参数（以及所有内部使用\$.ajax()函数的Ajax函数）。

为了完善Ajax的工具集合，jQuery也允许通过触发不同阶段的事件来监控Ajax请求，允许为这些事件建立不同的处理器。可以使用on()方法来绑定处理器，或者使用更方便的方法：ajaxStart()、ajaxSend()、ajaxSuccess()、ajaxError()、ajaxComplete()、ajaxStop()。

多亏这些强大的Ajax工具库，才可以轻易实现Web应用程序的丰富功能。记住这些知识，下面深入学习真实的项目demo。

为什么使用服务端验证？

在Web应用中，为什么使用服务端验证而不是客户端验证？客户端验证（如JavaScript）的好处是它可以在用户提交数据之前进行验证，从而减少服务器的负担。但是，客户端验证也有缺点：它依赖于用户的浏览器，而浏览器可能会禁用JavaScript，或者用户可能会使用不同的浏览器。因此，服务端验证是更可靠的选择，因为它可以在服务器端进行验证，无论用户是否使用JavaScript，或者使用哪种浏览器。

图11-21 显示了一个包含多个错误的示例。

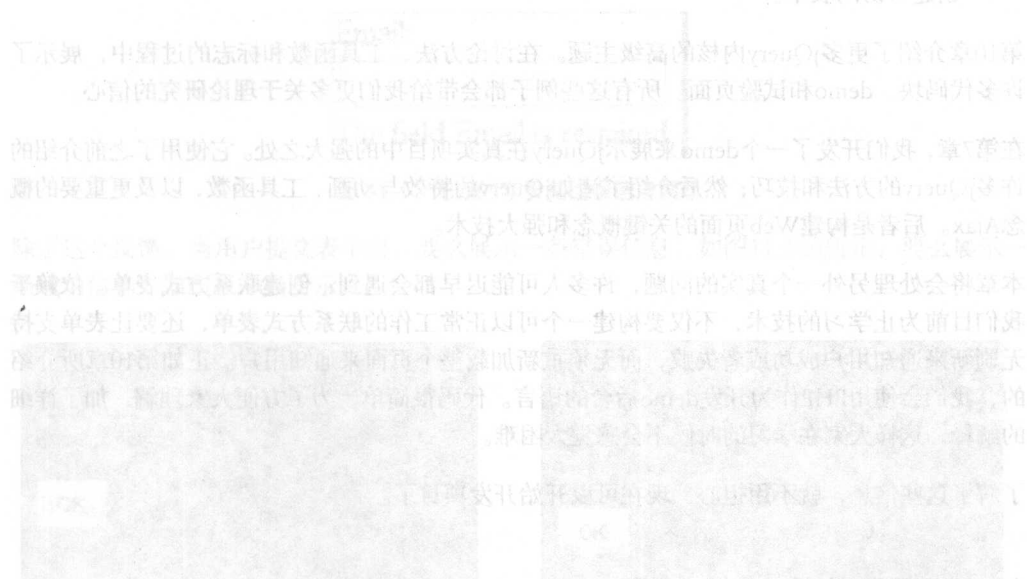


图11-21 显示了一个包含多个错误的示例。图11-21 显示了一个包含多个错误的示例。图11-21 显示了一个包含多个错误的示例。

图11-21 显示了一个包含多个错误的示例。图11-21 显示了一个包含多个错误的示例。图11-21 显示了一个包含多个错误的示例。

图11-21 显示了一个包含多个错误的示例。图11-21 显示了一个包含多个错误的示例。图11-21 显示了一个包含多个错误的示例。

demo:Ajax驱动的联系方式表单

Demo:an Ajax-powered contact form

本章内容

- 使用jQuery特效。
- jQuery工具函数。
- 发送Ajax请求。
- 创建可访问表单。

第10章介绍了更多jQuery内核的高级主题。在讨论方法、工具函数和标志的过程中，展示了许多代码块、demo和试验页面。所有这些例子都会带给我们更多关于理论研究的信心。

在第7章，我们开发了一个demo来展示jQuery在真实项目中的强大之处。它使用了之前介绍的许多jQuery的方法和技巧；然后介绍了诸如jQuery的特效与动画、工具函数，以及更重要的概念Ajax。后者是构建Web页面的关键概念和强大技术。

[301] 本章将会处理另外一个真实的问题，许多人可能迟早都会遇到：创建联系方式表单。依赖于我们目前为止学习的技术，不仅要构建一个可以正常工作的联系方式表单，还要让表单支持无刷新来通知用户成功或者失败，而无须重新加载整个页面来通知用户。正如第10章所介绍的，我们会使用PHP作为开发demo后台的语言。代码很简单，为了方便大家理解，加了详细的解释，这样大家在学习的时候不会感觉太困难。

了解了这些信息，就不用担心，现在可以开始开发项目了。

11.1 项目功能

The features of the project

在开发项目之前，先来讨论一下项目的需求。demo只需要两个页面：一个是表单（index.html），另外一个处理后台的业务逻辑（contact.php）。可以在chapter-11文件夹下找到例子页面代码。

为了简化需求，并且保持趣味性，需要创建包含四个字段的表单：全名、email、主题和消息内容。这些字段必须填写信息。最后，email地址必须是正确的格式，其他字段至少有四

个字符。

表单是高交互性的，而且会检查字段是否符合约束条件，而不需要重新加载页面。为此，需要使用之前学习的技术，特别是Ajax相关的知识，来发送请求给服务器。

当用户点击“Submit”（提交）按钮时，都会验证用户的输入是否有效。此外，还会在字段控件失去焦点的时候进行单独验证。如果无效，就会通过信息警告用户输入的消息错误。

为什么使用服务端验证？

可能有些人好奇，为什么使用服务端验证而不在客户端使用JavaScript进行数据验证。原因是每次使用JavaScript执行的验证都是不可靠的和不安全的，因为用户可以轻易禁用JavaScript。因此，不要允许用户发送无效的和有潜在危险的数据，或者在服务端重复相同的验证。为了避免这些问题，我们使用服务端验证，并且使用Ajax保留页面交互性。

图11.1展示了描述的例子功能。

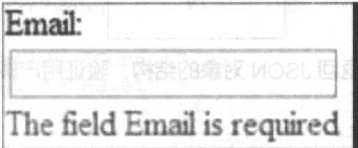


图 11.1 无效的字段和错误信息显示

除了这个反馈，当用户提交表单时，要么展示一条错误信息，如图11.2(a)所示，要么展示一条成功信息，如图11.2(b)所示。

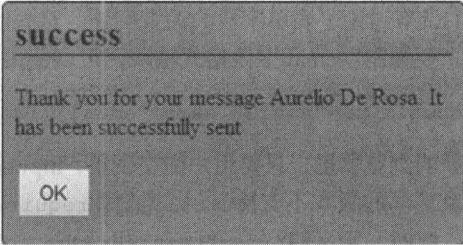
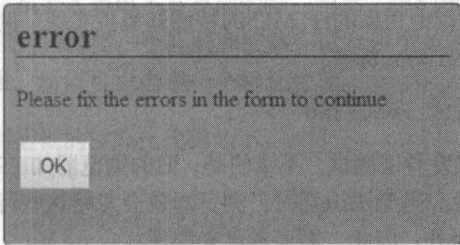


图 11.2(a) 当表单出现一个或者多个错误时的提示对话框

图 11.2(b) 当表单字段有效并且成功发送消息时的提示对话框。在成功消息里包含发送者的名字（此时是 Aurelio De Rosa）

PHP页面返回的消息是JSON对象，而且会使用jQuery将其注入对话框中。返回JSON的结构如图11.3所示。

JSON对象的结构很简单，由三个属性组成：status、message和info.status。info.status

是一个字符串，用来指定用户输入的值是否正确。如果是，就为success；否则为error。message包含要在对话框中显示的字符串。在验证单个字段的时候，如果后者是有效的，则message为空。info属性提供了一个数组，包含与无效表单字段相关的信息。每个对象都包含两个属性：field和message。field用于指定错误字段的名称。message与之前介绍的作用一样，但是这次它用来提供专门字段的错误。

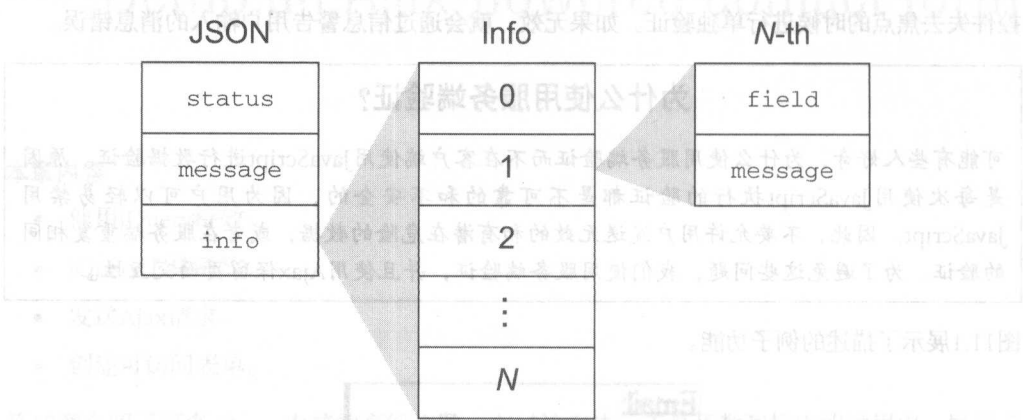


图 11.3 PHP 页面返回 JSON 对象的结构，验证用户输入并且发送 email

继续下一步：表单的HTML标签。

11.2 创建页面标签

Creating the markup

第11.1节讨论了表单字段的约束问题。这些约束可以使用新的HTML5的特性和类型实现。例如，要强制以email字段格式输入，可以使用下面的input标签：

```
<input type="email" name="email" id="email" required />
```

在新的浏览器里，只要用户提交表单，input就会触发验证测试。不幸的是，旧的浏览器如IE9以及更早的版本不支持email类型和required特性。当无法识别时，浏览器就会忽略这些行为。此时，定义的type="text"和required没有用处。因此，需要自己来实现。

如果想在旧的浏览器上兼容这些新的HTML5特性和类型，则可以使用第4.1节学习的技巧：

```
if (!('required' in document.createElement('input'))) {  
    //设置自己的控制  
}
```

在这个demo里，尽可能保持简单，假设已经忘记了HTML5。要自己控制，不使用HTML特性。我们来看看表单的代码，如列表11.1所示。

列表 11.1 联系方式表单的 HTML 标签代码

```
<form id="contact-form" name="contact-form" class="box" method="post"
  action="contact.php">
  <div class="form-field">
    <label for="name">Full name:</label>
    <input name="name" id="name" />
    <span class="error"></span>
  </div>
  <div class="form-field">
    <label for="email">Email:</label>
    <input name="email" id="email" />
    <span class="error"></span>
  </div>
  <div class="form-field">
    <label for="subject">Subject:</label>
    <input name="subject" id="subject" />
    <span class="error"></span>
  </div>
  <div class="form-field">
    <label for="message">Message:</label>
    <textarea name="message" id="message"></textarea>
    <span class="error"></span>
  </div>
  <input type="submit" value="Submit"/>
  <input type="reset" value="Reset"/>
</form>
```

正如我们从列表中看到的，每个字段都由三个元素组成：label、input（textarea用于存储消息）、span。label用来指定字段的名字，for属性用来改进表单的可访问性。<input>和<textarea>允许用户输入信息。span用来展示图11.1所示的反馈。此外，除了这些元素，还有Submit（提交）和Reset（重置）按钮。

联系方式表单不是页面的唯一组件，我们也需要一个展示信息的对话框。后者的代码很简单，因为只需要一个标题和段落，以及一个关闭对话框的按钮即可。这三个元素封装在一个容器里，所以可以当成一个组件处理。

对话框的HTML代码如下：

```
<div class="dialog-box">
  <h2 class="title"></h2>
  <p class="message"></p>
  <button>OK</button>
</div>
```

到此对话框，我们已经学习了index.html页面的全部HTML代码。如果在当前状态下运行这个页面，它既不是交互式的，也不是可用的，因为它没有做任何事情。

可以通过添加后端代码来解决这个问题。如果不熟悉PHP也不要担心，我们会高亮强调关键知识点。

11.3 实现 PHP 后台

Implementing the PHP backend

后台页面有两个职责：验证用户输入和发送邮件。前者最有意思，因为根据项目的规格需求，要处理两种情况。第一种是部分请求，当某个字段失去焦点时。第二种是发送包含所有字段的值的请求，当用户点击“Submit”（提交）按钮时。

为了区分这两种情况，添加了自定义参数`partial`来标志部分请求（在JavaScript代码中会详细讨论此参数）。

此时，PHP页面必须跳过所有的字段验证只处理这个特殊的字段。然后返回包含验证结果的JSON对象。结果会保存在一个叫`$result`的变量里，并且使用PHP函数`json_encode()`返回（使用echo语言构造），代码如下：

305 `echo json_encode($result);`

为了区分部分和完整请求，可以使用名为`$isPartial`的变量，它的值如下：

```
$isPartial = empty($_POST['partial']) ? false : true;
```

为了帮助大家理解后台代码，我们分析与`name`字段相关的验证代码，如列表11.2所示。

列表 11.2 验证 `name` 字段的代码

```
if (!$isPartial && $_POST['name'] === '') {
    $result['info'][] = array(
        'field' => 'name',
        'message' => sprintf($messages['required'], 'Full name')
    );
} else if ((!$isPartial || isset($_POST['name']))
    && strlen($_POST['name']) <= 3) {
    $result['info'][] = array(
        'field' => 'name',
        'message' => sprintf($messages['short'], 'Full name', 4)
    );
}
```

① 请求不是部分，而且值是空

② 设置恰当的
错误数据

③ 如果请求不是部分或者名字已经输入，但是它少于4个字符

④ 设置恰当的
错误数据

这段代码最有意思的部分就是两个if条件，第一个测试请求是否为部分请求（使用变量`$isPartial`）和值是否为空（`$_POST['name'] === ''`）①。如果都为true，就会设置适当的错误消息，提示值是强制的，必须填写②。

第二个if③有点复杂。我们想要验证输入的值是否少于4个字符（`strlen($_POST['name']) <= 3`），并且如果是，则返回错误消息。但是当执行验证时，事情变得有意思了。如果请求不是部分请求（`!$isPartial`），则必须检验长度；或者如果是部分请求，则只关注`name`字段（`$isPartial && isset($_POST['name'])`）。根据讨论，也许认为最终的条件是这样的：

```
!$isPartial || ($isPartial && isset($_POST['name']))
```

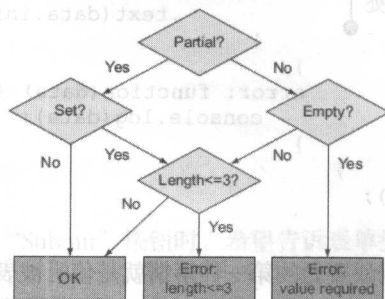
如果左边为false，则只有OR操作符的右侧会被评估——当请求是部分请求的时候。依赖于这个信息，可以缩短条件，代码如下：

```
!$isPartial || isset($_POST['name'])
```

如果条件为true，则设置适当的错误消息❶。

理解这个条件可能有点困难，但是自行阅读代码几次，就会发现我们写的是对的。假如你还不明白，可以看一下如图11.4所示的流程图。

验证其他字段的代码类似，这里不再重复讨论了。



306

图 11.4 验证用户输入的处理流程

既然已经学习了后台代码，现在来看看项目最有意

思的部分：前台JavaScript代码。第11.4节会介绍如何使用jQuery编写代码，让页面活起来。

11.4 使用 Ajax 验证字段

Field validation using Ajax

表单已经准备就绪，而且还有一个PHP页面来处理请求。老的方式，同步世界里，这个已经够了。用户填写表单，然后点击“Submit”按钮发送数据给服务器；后者使用服务端语言来处理数据、验证数据，最后生成输出页面。今天，用户希望网站具备高交互性，而且可以使用Ajax满足这个需求。

第一个要开发的功能就是给用户反馈输入字段的有效性，而不需要提交整个表单。这是不错的提升；它节约了用户的流量，因为不需要重新加载所有的页面数据。

想法是希望当字段控件失去焦点的时候，尽快给用户反馈结果。要实现这个功能，需执行Ajax请求，发送要验证的字段值。如果无效，则必须显示服务器的错误消息。假设请求发生问题，那么希望记录日志到控制台以便于调试项目。实现的代码如列表11.3所示。

列表 11.3 当用户输入数据的时候进行验证

```

$('input, textarea', '#contact-form').blur(function() {
    var $this = $(this);

    $.ajax({
        'contact.php',
        {
            method: 'POST',
            dataType: 'json',
            data: $this.serialize() + '&partial=true',
        }
    })
  
```

❶ 为 input 和 textarea 元素的 blur 事件附加处理器

❷ 执行 Ajax 请求

❸ 发送数据的页面

❹ 指定使用 HTTP 的 POST 方法

❺ 把字段作为额外参数序列化到请求消息体中

❻ 设置期望接受的数据格式

307

```

定义当请求成功返回结果后执行的处理器
success: function(data) {
    if (data.status === 'error') {
        $this
            .next('.error')
            .text(data.info[0].message);
    }
},
error: function(data) {
    console.log(data);
}
);
});

```

⑤ 设置错误消息

⑧ 定义出错时的回调函数

代码块要做的第一件事情就是侦听被表单的input或textarea触发的blur事件①。在这个事件处理器内部，使用\$.ajax()工具函数向服务器发送数据，因为我们希望对请求消息进行更多的控制②。

此函数的第一个参数是接收数据的页面contact.php③。除了这个页面，还传递一个对象作为第二个参数，该对象用于查询定义请求。根据之前学习的知识，如果想使用HTTP请求方法，就必须使用POST请求。可以把method属性设置为POST④。第二个属性是dataType。我们知道，PHP页面会返回JSON对象，因此赋值为json⑤。我们还需要序列化字段和值，添加特殊的参数(partial=true)来标记请求是部分请求⑥。一旦所有这些属性设置完毕，就可以开始为两个主要的状态建立回调函数：success和error。

在success回调函数⑦里，要检查返回对象的status属性。如果是error，就需要设置下一个span元素（第11.2中介绍的）的值为第一个info属性的值⑧。在error回调函数里，它表示错误请求，要在控制台记录日志信息⑨。

有了这些代码，就可以通知用户输入数据的有效性了。一旦某个字段出错，用户可能需要返回字段控件来修改数据。当字段查询聚焦的时候，要隐藏用户信息，这样以免误导用户新数据也是错的。实现这个功能十分简单。我们要做的就是将focus事件的处理程序设置给选择的元素。在处理器内部，隐藏了span元素。因为必须操作在相同集合的元素上，所以可以通过列表11.3的链式代码实现这个功能来节约代码量。相关的代码修改如下：

```

$('input, textarea', '#contact-form').blur(function() {
    //这里忽略代码...
})
.focus(function() {
    $(this)
        .next('.error')
        .text('');
});

```

在这个代码块和列表11.3中，使用了之前学习的一些方法。使用blur()来附加给blur事件，使用\$.ajax()工具函数来发送异步请求给服务器。serialize()可以帮助我们获取字段控件的名称和值，并转换为字符串。next()方法允许我们查询字段右边的span元素。最后使用text()方法设置span元素的文本。这些就是如何使用jQuery方法和工具函数来为网站页面创

建功能的例子。

不论输入值是否有效，用户仍然点击“Submit”按钮。此时，动作会触发经典的同步请求，这是要避免的。下面看看如何改变默认的行为。

11.5 Ajax 更多乐趣

Even more fun with Ajax

快速响应用户是一个不错的功能。然而，当用户点击“Submit”按钮时，希望告诉表单接下来怎么做。

要实现这个功能，需要附加事件处理器给submit事件。此处理器的目标就是向contact.php执行Ajax请求，发送输入的验证数据。只要服务端返回结果，就要分析它。如果包含错误，则需要在每个字段的span展示错误消息。然后展示包含通用请求信息的对话框，表示成功或者失败。这个功能的代码如列表11.4所示。

列表 11.4 管理通过 Ajax 提交请求的事件处理器

阻止默认
的动作

添加表单 submit
事件的处理器

发送异步请求

设置每个字段
的错误消息

设置对话框的
标题和信息

停止当前动画并显示对话框

```
$('#contact-form').submit(function(event) {  
    event.preventDefault();  
  
    $.post(  
        'contact.php',  
        $(this).serialize(),  
        function(data) {  
            if (data.status === 'error') {  
                $.each(data.info, function(index, elem) {  
                    $('#'+elem.field)  
                        .next('.error')  
                        .text(elem.message);  
                });  
            }  
        },  
        'json'  
    );  
    var $dialogBox = $('#.dialog-box');  
    $dialogBox  
        .children('.title')  
        .text(data.status);  
    $dialogBox  
        .children('.message')  
        .text(data.message);  
    $dialogBox  
        .finish()  
        .show();  
});
```

这些代码和第11.4节展示的代码不同。这里为表单的提交事件建立了一个处理器❶。在它内部阻止了默认的操作，即发送同步的POST请求——使用`preventDefault()`方法阻止。这是因为我们不想POST请求在执行代码后发送出去❷。

下一个要执行的操作就是发送请求给服务器。再说一次，需要发送POST请求。这时不需要为错误状态指定回调函数，所以可以使用`$.post()`工具函数，因为它的参数足够满足需求❸。第一个参数是接收数据的页面`contact.php`。然后序列化表单，这样字段的值就会包含在请求消息体中。第三个参数是成功时的回调函数。最后，指定期望的数据类型(`json`)。

在处理`$.post()`成功处理器中，检查返回对象的状态值。如果错误，就迭代每个字段的错误信息属性。

不论字段的有效性，还必须展示对话框来显示反馈信息。为此，设置标题和对话框信息❹，然后展示它们❺。在显示对话框前，要确保之前的动画都停止，并调用jQuery的`finish()`方法。

有了这些代码，用户点击“Submit”按钮，处理器就会执行。显示对话框，报告成功或者失败消息。此刻，交互会停止。你知道为什么吗？花点时间思考下。

❸10 原因是当用户点击时没有关闭(隐藏)对话框。现在来修改这个bug。

11.5.1 隐藏对话框

关闭对话框对我们来说应该不难。但是在深入代码之前，先来做一个小的优化。

在列表11.4所示的Ajax请求处理器中，代码定义了变量`$dialogBox`：

```
var $dialogBox = $('#dialog-box');
```

因为要再次操作页面上的这个元素，所以可以减少敲击键盘，通过移动这行代码到开始处来优化性能（在`script`元素顶部）。

要点击按钮时，隐藏对话框，需要为按钮的`click`事件添加侦听器。这可以通过下面的代码实现：

```
$dialogBox.children('button').click(function() {
    $(this)
        .parent()
        .hide();
});
```

有了这些附加的演示代码，就满足了当初的设计需求。如果对于编写如此简洁的代码感到兴奋，想在下一个项目里使用联系方式表单页面的代码，那没有问题。

对于要深入学习jQuery开发的读者，下面看看如何加入一些特效来完善项目。

11.6 使用动画特效改善用户体验

Improving the user experience using effects

动画和特效从来都不是网站应用程序必须的部分，有没有它们，用户都应该可以正常执行任务。但是，某些情况下，它们可以改善用户体验。在这个项目里，可以允许错误消息和对话框逐步出现和显示，而不是立即出现或者消失。

要添加的第一个特效就是对话框的按钮处理器。可以更新代码，这样对话框就可以慢慢隐藏了。可以使用jQuery的`slideUp()`方法来替换`hide()`方法。不需要传递任何参数，动画特效会持续默认400毫秒。新的代码如下：

```
$dialogBox.children('button').click(function() {
    $(this)
        .parent()
        .slideUp();
});
```

如果要控制动画的持续时间，那么可以传递任意的时间参数。

◀ 311

使用隐藏对话框相同的动画代码处理方式，可以修改展示的代码。可以使用`slideDown()`方法，但是为了差异性，也可以使用带参数的`show()`方法。正如第8.2.1节所述，可以传递表示毫秒的参数，也可以使用字符串参数`slow`、`normal`、`fast`控制。如果不想用户在对话框显示之前等待太久，就传递`fast`字符串，它会持续200毫秒。

其他的特效也可以添加给错误页面，但是把这个留给大家作为简单的练习。希望大家独立完成。

动画特效可能会取悦某些用户，也可能会招人讨厌。在为最广泛的用户提供最棒的用户体验的尝试中，我们要记住许多不同的观点。下面看看如何满足这些不想允许运行动画的用户需求。

11.6.1 卷缩特效

在第8章介绍了jQuery的标志（flag）。其中介绍的`fx.off`标志，它允许禁用全局的动画。为了给用户这个机会，需要提供给他们要使用的HTML元素。在demo里使用了有两个选项的下拉框——On和Off，你可以使用任意的HTML满足这个需求，比如复选框或者两个单选按钮。

select元素的代码如下：

```
<div class="animations-box">
  <label for="animations">Animations are:</label>
  <select id="animations">
    <option value="true" selected>On</option>
    <option value="false">Off</option>
```

```
</select>
</div>
```

我们把这些代码放到表单的上面。一旦完成，就只需要为select元素添加实际的处理逻辑代码。为此，需要侦听下拉框的change事件，当用户选择不同的选项时，更新fx.off标志。下面就是实际的代码：

```
$('#animations').change(function() {
    $.fx.off = $(this).val() === 'false';
})
.change();
```

正如我们看到的，不仅侦听了change事件，还在设置处理器之后触发了它。这确保了标志设置为默认的<select>值。这些代码在想要选择Off选项关闭动画的时候就会起作用，因为不需要更新JavaScript代码(默认fx.off标志位为true)。

312 在结束这个项目之前，还有最后一点需要讨论。

11.7 注意访问性

A note on accessibility

JavaScript是一门强大且广泛应用的语言，允许执行许多不同的任务。正如在本书中看到的，jQuery提供了更加简化的开发模式，几行代码就可以实现复杂的功能。当开发Web网站时，记住，不是所有人都可以加载并执行JavaScript代码的。有些用户的计算机可能已经禁用了JavaScript，或者服务器无法提供JavaScript库、模块或者通用文件。此时，作为Web专业开发人员，必须有自己的备用计划。

在这个项目中，使用了JavaScript来增强用户体验。如果加载JavaScript代码失败，那么联系方式表单仍然可以正常工作。Submit按钮将会发送表单数据给服务器。这存在可能性，因为开发代码是建立在原生的HTML之上的。唯一的缺点是过程会变成同步模型，用户会看到一个新页面（或者相同的页面包含不同的内容）。但这是真的吗？用户实际上真的会看到页面吗？

我们开发的contact.php页面只提供JSON数据对象。如果JavaScript加载失败，用户提交表单，用户看到的的就是包含JSON数据的原始对象，难以阅读，难以理解。耻辱！这是我们能力的极限吗？

正如所证明的，可以微微修改项目来解决这个问题，我们要做的就是编辑contact.php页面，这样它就可以区分请求是否是Ajax请求。如果是Ajax请求，它就提供JSON对象；否则使用数据生成新的页面来服务用户。虽然看起来是一个简单的修改，但是它改进了用户体验，并且避免了误导用户。这个改进方法叫持续增强（progressive enhancement）。

持续增强

持续增强是一种方法，它强调可访问性、语义HTML标签，以及外部样式和脚本技术。这个词被Steven Champeon在一系列文章，以及2003年的Webmonkey和SXSW交互性会议中提出。

此方法增强了网页的创建方式，这样每个用户都可以访问基本内容和功能，为使用更好的技术（例如，新的浏览器）提供一个增强版本。它不仅改善了网页的访问性，也改进了页面在SERP（search engine result pages，搜索引擎结果页面）中的排名，所以应该在未来的项目中采用这个技术。

◀ 313

JavaScript失败的处理计划并非改进网页访问性的唯一方式，也可以采用WAI-ARIA (<http://www.w3.org/TR/waiaria/>)访问。虽然详细介绍已经超出了本书的范畴，但是用户可以有一个概念认知，它提供了一个论点，基于角色、状态和属性来定义用户接口元素的可访问性。这些改进了Web内容和程序的访问性与互操作性。暴露的角色之一，dialog就是满足对话框需求的重要元素。要在项目中使用，必须添加属性给HTML页面元素，代码如下：

```
<div class="dialog-box" role="dialog" aria-labelledby="dialog-title"
    aria-describedby="dialog-desc">
  <h2 id="dialog-title" class="title"></h2>
  <p id="dialog-desc" class="message"></p>
  <button>OK</button>
</div>
```

另外一个可行的改进就是使用HTML5的required属性，我们在本章开篇的时候介绍过。使用它可以支持HTML5的工具来强制用户输入数据到字段中。辅助技术（assistive technologies, ATs）的处理速度较慢（如Chrome、Firefox等），为了弥补这个不足，可以使用WAI-ARIA的aria-required属性来强制元素，代码如下：

```
<input name="name" id="name" required aria-required="true" />
```

即使UA支持HTML5，添加一些WAI-ARIA特性也不会有问题。

这里讨论的增强技术，是可以用来改进联系方式表单的一个小集合。但是这些小修改练习足以让大家认识到网页开发中的增强访问性问题，能为后续的项目做好准备。

11.8 总结

Summary

本章通过开发一个简单但是功能完整的Ajax联系方式表单来把所有的知识融合到一起。在开发demo的过程中，介绍了本书中涉及的许多主题。使用了选择器，包括context参数，它在第2章中介绍过。像parent()、next()、find()，这在第3章里介绍过，用来精简元素选择代

码。text()方法，这是第5章里介绍的，用来更新包含错误消息的span元素的文本。还使用了一些与事件有关的方法，第6章讨论的，用来侦听事件（比如blur()、click()、focus()）。还添加了一些动画特效，第8章介绍的，让元素逐渐出现。第9章中介绍的\$.each()工具函数，可帮助我们迭代错误数组元素。最后，使用了第10章介绍的\$.ajax()和\$.post()工具函数来执行异步请求。

这个例子应该包含目前为止介绍的许多概念知识点，它们不仅在理论上可行，而且在实际的网站中经常会用到。在这个demo中，至少使用了每一章中介绍的一个概念。

314 我们希望通过本章的这些页面开发，可以了解jQuery库的每个部分对于实现某个目标的重要性，以及如何把这些知识组合起来发挥jQuery的强大威力。希望大家喜欢上这个项目的开发过程，并且对于讨论的知识更加明白、自信。

这个例子完成以后，我们就结束了本书的第二大部分内容。后面将深入学习一些更高级的主题，比如创建jQuery自定义插件库和单元测试代码。

315

```
div class="dialog-box" role="dialog" aria-labelledby="dialog-title" style="display: none;">
  

### 对话框标题



这里是对话框的正文内容。



底部内容


```

总结 8.11

第三部分

高级主题

Advanced topics

本章内容

• 为什么jQuery不是足够强大...

• 使用jQuery扩展库

• 如何扩展jQuery核心

在本书的第二部分，我们介绍了许多强大的jQuery选择器、方法以及工具函数。如果大家已经掌握了这些知识，现在就可以创建任意想要的功能了。在最后一章里，我们有了足够的知识，一切皆有可能，唯一的限制就是你的想象力。

虽然jQuery很强大，但是它并不包含满足各种需求的方法或者函数。为了弥补这个不足，jQuery可以很方便地进行扩展，允许Web开发者引入自己的函数作为jQuery内核的一部分。第12章将会学习如何创建jQuery插件库。然后讨论Deferred对象及其方法。Deferred对象属于jQuery内核，我们选择了单独介绍，因为知识点复杂，所以学习起来比较困难。

除非正在开发的是一个很小的项目——唯一使用的东西——将要编写的代码，会被重构、更新及修改。对于这种情况，想确保所有代码在修改之前和之后都能正常工作。一种方式就是测试项目。因为我们讨论到测试，为什么不采用相同的强壮的单元测试框架——QUnit？它由jQuery团队开发，用来测试jQuery库。听起来很合理，不是吗？这就是我们要在第14章里介绍的知识。

最后，在本书的最后一章，会介绍一些在开发大型项目中非常有用的工具、技巧、提示、窍门，并展示如何在项目中使用它们。

◀ 317

所有提到的主题都会区分那些只会使用jQuery的初中级工程师和那些能够在开发功能的时候有能力改进并优化代码的高级工程师，没有忘记开发质量保证代码（测试）。

我们会在后面的章节里学习这些知识。

为了不浪费大家的宝贵时间，下面开始深入学习这些jQuery高级主题。

◀ 318

jQuery 扩展插件

When jQuery is not enough...
plugins to the rescue!

本章内容

- 为什么要使用自定义代码扩展jQuery。
- 使用第三方插件。
- 高效扩展jQuery插件指南。
- 编写自定义工具函数。
- 为jQuery对象编写自定义方法。

在本书的学习过程中，我们看到jQuery不仅提供了大量有用的工具方法和函数集，而且可以与页面行为紧密结合。有时，代码后面隐藏着公共模式，我们想重复使用多次。当考虑这种模式时，有必要把重复代码封装成重复使用的工具添加到原始的工具集中。本章将会学习如何提取这些重复使用的代码块来作为jQuery的扩展部分，称为jQuery插件（jQuery plugins）。jQuery插件有两种模式：jQuery方法（如`find()`或`animate()`）或者工具函数（如`$.grep()`和`$.extend()`）。本章会介绍这两种模式。

319

当开发项目时，不可能自己全部开发自己需要的代码，特别是其他人已经开发完成的代码库。因此，也会介绍一些大家想学习的流行的插件。

但是在这之前，首先要讨论下为什么需要模式化封装自己的代码，以作为jQuery扩展库使用。

12.1 为什么扩展 jQuery

Why extend jQuery?

如果大家在阅读本书的过程中稍加留意，就会注意到采用jQuery以后页面脚本代码的开发方式发生了很大变化。

jQuery提升了页面代码的开发风格：先格式化jQuery集合，然后将jQuery方法或者链式方法应用到集合上。当编写代码时，虽然可以随心所欲，但是有经验的开发者都认同，所有网站的

代码或者至少主要的代码，应坚持一种连续性的代码风格，这是一条好的实践开发原则，而且也是推荐的原则之一。作为jQuery扩展插件封装代码的一个好处就是在整个网站内维护统一的代码风格。另外一个好处就是通过创建可复用组件，以后的项目会从中获利，而且其他开发者也可以从发布的代码中获利。

最后一个好处（其他人可能列举的更多）就是通过扩展jQuery插件，有更多的jQuery库可以使用。例如，通过创建新的jQuery方法，就会自动继承jQuery强大的选择器机制，以及跨浏览器的兼容性功能。当可以利用如此强大工具库的时候，为什么还要完全从零开始编写所有的代码呢？

基于这些，显而易见，通过扩展jQuery插件来编写可复用组件是非常聪明的方式。在学习本章的过程中，将会详细检查每个指南和模式，它们可以指导我们创建jQuery插件，并实战编写一些自己的代码。

在开始学习如何开发自定义扩展插件之前，先看看如何查找、发现、评价以及使用其他人开发的插件。

12.2 在哪里查找插件

Where to find plugins

经过几个月的艰苦工作之后，2013年1月16号，jQuery团队在官方博客里宣布了新发布的jQuery注册方式(<http://blog.jquery.com/2013/01/16/announcing-the-jquery-plugin-registry/>)。新的注册地址（<http://plugins.jquery.com/>）取代旧的、包含许多问题的注册地址。但是之后的两年时间，新的注册地址一直是只读模式，意味着新插件发布无法进行。作为新注册地址的替代方法，jQuery团队推荐使用npm (<https://www.npmjs.com/>)。

在这个URL地址中可以看到一个简洁的页面，有一个搜索框可以用来查找需要的插件。一旦返回搜索结果，就可以点击名字来查看插件的详细信息，包括下载地址。

虽然npm是推荐查找插件的渠道，但它不是唯一的渠道。另外一个带有漂亮UI的网站就是Unheap(<http://www.unheap.com/>)，它提供了更多的插件方式。

如果两个网站都无法满足需求，则请记住Google是我们的朋友。但是也请记住要验证源代码的有效性。毕竟，要在网站中使用这些JavaScript代码！还不满意？下面几节会告诉大家如何创建自定义插件。

知道在哪里查找jQuery扩展插件还不够。我们可不想自己的好网站插入坏的代码，或者更糟糕的是，错误开发的插件导致网站性能下降。第12.1.1节会介绍如何使用插件。

12.2.1 如何使用插件

开发项目时，使用第三方插件是聪明的方法，可以节约大量时间，因为没有必要重复开发、测试和维护同样的代码。但是这一直都对吗？以我们的经验来看未必。

添加插件到项目中意味着添加了依赖于我们的网站。选择插件是一个重要的决策，因为整个项目都要基于它来构建。在实际应用某个插件之前，应该检查几个因素。某些因素严格依赖这些代码而其他因素是外部的。这些组合给了我们有关组件的稳定性和质量因素的提示。下面开始分析一些代码之外的因素。

代码之外的因素

使用第三方组件可从痛苦的底层开发任务中解放自己。但是，如果不注意组件的使用，就会发现自己浪费更多的时间在修改存在的bug以及了解如何使用插件上。虽然我们讨论的是jQuery插件，但请记住这几点也适用于工具、库和其他第三方软件。

注意：关于此问题的更深入的讨论，建议大家阅读 Nicholas Zakas 编写的书籍《New Perspectives on Web Design》（《Web 设计的新观点》，2013, <https://shop.smashingmagazine.com/smashingbook-4-new-perspectives-on-web-design.html>），该书 中的“Writing Maintainable, Future-Friendly Code”章节有详细介绍。

首先，要查看的就是插件的最后更新时间，它会告诉你作者对于此项目的关注度。长期不更新的插件基本意味着废弃的插件，这正是我们要避免的。在使用它之前，花费时间检查下它的修改日志。最后一次更新并非总是一个好的标准，因为每个为特定的jQuery版本构建的插件都会兼容一些其他的版本或者分支（更多信息请查看网址：<http://semver.org/>），所以大部分情况是向后兼容的。当某个更新破坏了插件时，会导致我们检查第二个因素。

321

第二个要考虑的因素是作者处理问题的速度。软件都不是完美的，一个问题可能有许多原因。作者修改bug的速度十分重要。如果插件停止工作是因为某个更新导致的，而我们又在使用它，那么在无法快速修复之前，就只能使用旧的jQuery版本或者自己修改bug了，而无法使用某些高级功能。

第三个重要因素就是插件的版本。除非是新手开发的，库的版本都有明确的含义（之前提供的链接里说明了）。因此，不要使用插件的0.1.0版本，除非为了乐趣。通常没有更新到1.0.0版本的软件都有许多修改，导致无法向前兼容。

第四个因素就是要检查作者。是公司还是个人开发者？公司或者个人开发者是否有很好的声誉？通常公司开发的组件会很好地进行维护，因为公司可以投入资金，而个人开发者通常只是花费业余时间。但是，即使作者是一个人，如果非常权威，且值得信任，那么插件也可以使用。

还有一个因素就是插件的文档。如果文档资料很差或者缺少文档，最好还是找新的替代品。这种扩展插件会强制我们花费更多的时间来了解其工作原理和使用方法。

作为结论，记住，不是所有的jQuery插件都有相同的质量，而且检查它们的质量也是我们的职责。

本节中分析的因素十分重要，但是仅仅是部分原则。要正确评估一个插件，也需要我们完全弄明白它的代码。

这一点并不是建议大家上网阅读要使用的插件的所有代码，这个实践原则要花费许多时间，要总览源码的质量、分析例子、标注坏的代码。为了能够准确评估源代码，需要学习创建插件的实践原则。因此，要求大家等到下面几个页面，介绍插件的创建过程，一步一步实战。

现在看看如何使用第三方插件。

使用插件

322 一旦发现插件满足需求，而且经检查确定值得我们花费时间，就可以添加到网站项目中。使用编写良好的插件通常很简单。所有要做的就是存储到Web服务器可以访问的文件夹中，再添加到jQuery库后面。

第一个例子，我们来看看插件jQuery Easing(<https://github.com/gdsmith/jquery.easing>)，这是第8.3节介绍easing函数时涉及的内容。一旦下载完成，就可以存储到网页可以访问的文件夹中。例如，可以存储在名为“javascript”的文件夹里。然后在页面jQuery库后使用script元素来添加引用。如果在jQuery之前添加，就会收到错误，而且所有网页中的JavaScript都会停止工作。网页中的代码类似如下：

```
<script src="javascript/jquery.1.11.3.min.js"></script>
<script src="javascript/jquery.easing.min.js"></script>
```

设置完这些标签后，下面发生什么情况取决于使用什么插件。在这个例子里，没有调用新的jQuery方法或者工具函数。jQuery Easing只会为jQuery核心库注入easing函数，允许我们使用它们。

这个插件是一个特例，因为许多插件需要在页面中添加新的标签或样式名或ID。实际的例子就是slick(<https://github.com/kenwheeler/slick>)，用来实现轮播效果。下面的例子会使用slick来创建图片轮播效果。

第一步，使用slick就是要在引用jQuery库后添加JavaScript文件，标签代码如下：

```
<script src="javascript/jquery.1.11.3.min.js"></script>
<script src="javascript/slick.min.js"></script>
```

在添加完JavaScript文件之后，还必须添加slick的CSS样式文件引用。正如第1章里学习的，JavaScript文件应该在</body>标签结束之前设置，而CSS文件应该放在<head>中。如果把CSS样式文件存储在名为“css”的文件夹中，代码设置应该如下：

```
<head>
  <link rel="stylesheet" href="css/slick.css" />
```

一旦完成,就应该设置页面的标签代码。因为要制作一个图片的轮播效果,必须使用一个容器元素(这里使用<div>)来包裹图片,代码如下:

```
<div class="carousel">
  
  
  
  
</div>
```

323

代码设置完毕,唯一剩下的步骤就是调用slick()方法来展示轮播效果,代码如下:

```
<script>
  $(''.carousel').slick();
</script>
```

这个语句依赖于默认的插件配置,但是也可以根据实际需要修改设置。如果要深入学习这个插件,可以查看存储库和官方文档。

通过这两个例子,大家应该对如何在页面中使用第三方插件有了完整的认识。现在,在开始创建自定义插件之前,来看一些比较流行的jQuery插件。

12.2.2 好插件

本节介绍一些最流行的和最经常使用的jQuery插件,可以在项目里执行公共的任务。这个列表没有完全列举所有的插件,但是足够我们学习使用。

第一个推荐的插件是typeahead.js(<https://github.com/twitter/typeahead.js>)。它是由Twitter开发的一个快速的、功能完整的自动完成插件。这意味着可以传递给它一个数据集合,它会在用户输入<input>的时候显示提示功能。

第二个值得一提的jQuery扩展插件是isotope(<https://github.com/metafizzy/isotope>)。它允许用户使用不同的动画效果来过滤和排序UI元素。例如行、列以及著名的masonry。

另外一个有意思的插件就是pickadate.js(<https://github.com/amsul/pickadate.js>)。这是一个移动友好、响应式、轻量级的jQuery日期和时间插件。它在<input>元素上添加了一个微件,所以用户聚焦到这个元素时,时间和日期就会显示出来,用户可以很方便地进行选择。

第四个插件是Chosen(<https://github.com/harvesthq/chosen>)。这个库是为了生成长的、不宽的下拉选择框,更加人性化而且漂亮。

velocity(<https://github.com/julianshapiro/velocity>)插件重新使用了jQuery的animate()方法来改善性能,并且引入了新的功能。

最后两个要推荐的插件是jCarousel (<https://github.com/jsor/jcarousel>)和Magnific Popup (<https://github.com/dimsemenov/Magnific-Popup>)。jCarousel插件用于实现轮播效果,不仅可

以实现图片轮播,还可以针对其他元素。Magnific Popup是一个轻量级、响应式的light-box(光合)脚本,关注在性能上。

324 这些插件已经变得非常流行,因为它们使用聪明的和高效的方式解决了一个真实的问题,或者比其他插件更早地解决了这些问题。无论什么原因,都希望自己的插件像这些列举的插件一样成功。为了实现这个目标,需要学习如何开发优秀的插件。这也是第12.3节内容最核心的目标。

12.3 jQuery 插件编写指南

The jQuery plugin authoring guidelines

本节包含一系列指南,帮助大家命名和构造插件。

这些指南不仅可以确保代码正确插入jQuery架构,还可以确保它与其他jQuery插件和JavaScript库一起正常工作。这里会列举当编写插件时的一些基本的和最好的实践原则。

扩展jQuery有两种形式。

- 操作jQuery集合的方法(我们称为jQuery方法)。
- `$`(jQuery的简称)上定义的工具函数。

在本节的其余部分会介绍一些常用的规则,然后会在某些章节里深入介绍某些特定的类型。

为了帮助大家学习,会在实战代码中演示这些规则。目标是构建Jqia Context Menu (Jqia 是jQuery in Action的简称),这个jQuery插件会为页面上的一个或者多个元素显示一个自定义上下文菜单。上下文菜单是当用户右击页面或者按下菜单键时在PC屏幕上显示的菜单。

对于上下文菜单,插件会使用页面的某个元素(典型的是列表)默认隐藏起来,因此要提前设置到页面中。作为菜单的页面元素,要通过ID查找。插件允许两个操作,所以它有两个方法:一是要初始化,二是要销毁动画特效。当初始化时,插件会重写默认的右击行为(展示通常的上下文菜单)来显示自定义菜单。当执行销毁动画特效操作时,插件会清空资源,然后恢复默认的行为。

最后,要做得更加出色,允许开发者来重写默认的左击行为。此时,无论鼠标怎么点击,都将显示自定义菜单。默认情况下,这个选项是禁用的。

既然已经介绍完了需求,现在是时候卷起袖子大干一场了,因为还有许多工作要做。

12.3.1 文件和函数命名规则

开发插件要做的第一个决定就是名字。当命名插件时,必须避免名字冲突。避免开发的插件

与其他文件或插件冲突非常重要，名字冲突对于Web开发者来说非常头痛。

jQuery团队推荐的原则简单而高效，提倡的原则如下。

- 插件名字简洁但意义明确。
- 文件名前缀使用jquery。
- 选择性添加公司名字和组织名字。
- 遵循插件的名称。
- 选择性添加插件版本。
- 文件结尾是.js。

拿Jqia Context Menu插件作为例子，如果要遵守这些推荐原则，文件名应该是jquery.jqia.contextMenu-1.0.0.js。

使用jquery前缀理论上可以消除与其他库文件名的冲突。毕竟，其他人编写的非jQuery插件没有必要使用jquery作为前缀，但是社区中关于插件的名字仍然是可以自由争论的。在我们的例子中，插件的名字由多个单词组成（此刻许多名字已经被占用了）。可以使用驼峰法则编写，也可以使用小写字母编写所有单词，甚至使用圆点或者短横线分割。使用哪种方法都行，根据自己的专业喜好选择，建议选择一个，然后一直遵守它。

一种保证插件名不会与别的插件冲突的方法就是以唯一的名字或者单位组织的名字作为后缀。例如，如果想要前缀属于本书的插件，可以使用文件名前缀jquery.jqia，正如前一个例子里的写法一样。

第三点是可选的好理由。假设某些开发者在使用我们开发的插件，一切正常，插件非常成功。我们想发布一个新版本，包含新的功能，那么问题来了。假设文件为jquery.jqia.contextMenu-1.1.0.js，那么，使用此插件的开发者不仅要更新JavaScript文件，还要修改版本的后缀。如果插件文件为jquery.jqia.contextMenu.js，则会更简单，只要使用评论来注释版本即可。这样做，开发者只需要替换JavaScript文件而不需要更新标签。

大家应该明白这些解释命名规则的原因，可以把这些规则实战用到自己的项目中。先忽略关于版本的规则，因此继续创建名为jquery.jqia.contextMenu.js的文件吧。

本节强调了文件命名的重要性，以及无法想象开发者如何在网站里使用这些文件。这种关注点同样存在于\$符号中。下面深入学习之。

12.3.2 认识\$

已经编写了相当多的新jQuery代码，也看到了使用\$替代jQuery编程的方便性。但是，当编写的代码需要给他人的页面使用时，就没有办法漠视不管了。作为插件的作者，没有办法知道

其他Web开发者是否会使用`$.noConflict()`函数(第9.2节里讨论的)来允许其他库使用`$`简称(最著名的就是Prototype)。可以使用jQuery来取代`$`简称,但是讨厌这种用法,还是喜欢使用`$`。

在第9.2节里,介绍了一种设计模式叫IIFE(immediately-invoked function expression,立即调用函数表达式),附录里有详细介绍。它的目的就是确保`$`在本地指引的是jQuery,而不影响其他页面。这种模式也可以用来定义jQuery插件,代码如下:

```
(function ($) {  
  //  
  // 插件定义在这里  
  //  
})(jQuery);
```

通过传递jQuery给参数为`$`的函数,就可以保证函数体内的`$`指向jQuery对象。现在可以愉快地使用`$`来编写核心的插件代码。

有了这个诀窍,打开jquery.jqia.contextMenu.js文件,把之前的代码块放进去(可以忽略注释部分)。

现在来看看另外一个编写插件的处理参数的指南。

12.3.3 搞定复杂参数列表

绝大部分插件倾向于简单问题,只需要一些参数。智能默认值可以在忽略参数的时候初始化,而且当某些可选参数忽略时,参数顺序可能有不同的意义。

jQuery的`on()`方法就是这种行为的一个很好的例子。如果忽略可选参数`data`,则函数侦听器通常作为第四个参数,而现在作为第三个参数;如果忽略`selector`参数,则可以把处理器作为第二个参数。JavaScript的动态特性允许我们编写这种灵活的代码,但是这类东西可能崩溃并且变得复杂(对于Web开发者和插件作者都是),当参数数量增多的时候。当许多参数被忽略时,就增加了插件崩溃的可能性。

思考如下签名的函数:

```
function complex(p1, p2, p3, p4, p5, p6, p7) {  
  //在这里编写代码...  
}
```

这个函数定义了七个参数。现在假设除了第一个参数,都是可选的。当可选参数忽略时,有太多可选参数无法智能推测调用者真实的意图。如果函数调用者只忽略尾部参数,这就不是问题,因为可选尾部参数可以作为`undefined`处理。但是,如果调用者想指定`p7`,那么默认的`p2`到`p6`呢?而且,如果某些忽略的参数接受相同的数据类型(禁止依赖于传递值的数据类型)呢?那么调用者需要为忽略的参数使用占位符,编写代码如下:

327

```
complex(valueA, null, null, null, null, null, valueB);
```

更恶心的写法还有：

```
complex(valueA, null, valueC, valueD, null, null, valueB);
```

使用这个函数的Web开发者被强迫仔细计算`null`参数的个数以及参数的顺序。此外，代码也难以阅读和理解。但是不允许调用者传递这么多参数，该怎么办？

再一次，JavaScript灵活的本性拯救了我们。解决这种混乱问题的模式出现在社区里——可选哈希参数（options hash）。使用这种模式，可选参数可以聚集到单个（single）参数里，使用JavaScript Object对象实例，它的属性名/值对用来作为可选参数。

使用这个方法，第一个例子可以修改为：

```
complex(valueA, {p7: valueB});
```

第二个例子可以修改为：

```
complex(valueA, {  
  p3: valueC,  
  p4: valueD,  
  p7: valueB  
});
```

非常非常棒！

不需要使用占位符`null`处理可选参数，也不需要计算参数。每个可选参数都非常方便标记，这样它就能准确地代表自己（使用更有意义的名字而不是`p1`到`p7`）。

注意：某些API遵循捆绑可选参数到单个options参数中的原则。其他的API捆绑完整的、必需的和可选的参数到单个对象里。推荐使用第二个方法，因为必需参数的数量可能随着时间的增加，所以这种解决方法更有远见。

虽然对于复杂函数的调用者来说这是一个巨大的优点，但是对于插件开发者来说呢？事实证明，我们已经看到，jQuery提供了简便的方式来处理这些可选参数，并合并到一起设置默认值。思考下一个函数例子，有一个必备参数和六个可选参数。新的函数签名如下：

```
complex(p1, options)
```

◀ 328

在这个函数内，可以使用`$.extend()`工具函数来合并六个可选参数，并且设置默认值。思考下面的代码：

```
function complex(p1, options) {  
  var settings = $.extend({  
    p2: defaultValue1,  
    p3: defaultValue2,  
    p4: defaultValue3,  
    p5: defaultValue4,  
    p6: defaultValue5,  
    p7: defaultValue6
```

```

    },
    options || {}
  );

  //函数的其余部分
}

```

通过合并开发者传输的options参数到单个对象中，开发者显示设置了settings变量属性的默认值。

提示：可以使用options引用实际的值，而不需要创建新的settings变量。这样会减少堆栈上的对象引用，但是仍使用清晰的方式。

在前面的代码里，可以使用|| {}来处理options对象为null或者undefined的情况，如果options参数评估为false，则它会提供一个空对象。这样的方式简单、友好。

我们在创建优美且精细的插件上又前进了一步，现在将其用到实际的项目中。回忆之前的项目——Jqia Context Menu插件，需要可选参数来控制鼠标点击时如何显示自定义菜单。除了这个选项，还需要指定菜单要显示的元素的ID。虽然只需要两个参数，一个是强制的参数，一个是可选参数，但是会传递单个对象给插件。原因是正如之前提到的，这种方法更加有远见。

看看具体的代码，它必须替换正在编写的JavaScript文件，结果如下：

```

(function($) {
  var defaults = {
    idMenu: null,
    bindLeftClick: false
  };
  329 > })(jQuery);

```

在这段代码里，定义了一个名为defaults的对象，包含一个可以指定菜单ID(idMenu)的属性及另外一个属性，表示左键点击事件是否应该被重写(bindLeftClick)。

代码中定义了一个包含两个属性的对象，其他没有什么特殊的地方。继续学习下面开发插件的指南。

12.3.4 统一命名空间

与我们看到的其他命名规则一样，无论这些函数是新的工具函数还是jQuery集合方法，都应确保函数名不会与其他使用的插件冲突。

当创建自用的插件时，通常都知道自己网站要使用的组件；除非项目特别庞大，一般都很容易避免命名冲突。但是创建公开使用的插件呢？最初创建的私用插件，证明非常不错，如果想将其公开给社区使用呢？

为了更好地理解这个概念，下面看看具体的例子。正如我们提到的，Jqia Context Menu插件需要两种方法：`init()`和`destroy()`。可以尝试在JavaScript文件里添加下面的代码：

```
var init = function(options) {  
    //这里编写代码...  
};  
var destroy = function() {  
    //这里编写代码...  
};
```

悲剧的是，这并非我们真正需要的。

使用IIFE模式最主要的一点就是创建一个环境，在IIFE内部声明的变量和函数不能被外部访问。这种行为在不想自己的`defaults`变量只有内部方法可见而不被外部插件知道时非常有用。在IIFE里需要一种方法来向外界暴露我们的方法。

创建操作jQuery集合的插件，并添加方法到jQuery集合，我们必须把它们赋值给`$.fn`属性。更新JavaScript文件，以符合这些原则指南，新的代码如列表12.1所示。

列表 12.1 第一版 Jqia Context Menu 插件

```
(function($) {  
    var defaults = {  
        idMenu: null,  
        bindLeftClick: false  
    };  
    $.fn.init = function() {  
        // 这里编写代码...  
    };  
    $.fn.destroy = function() {  
        // 这里编写代码...  
    };  
})(jQuery);
```

330

这里展示的代码什么也没做，因为还没有定义`init()`和`destroy()`的方法体。尽管如此，也可以调用了。

新玩具总是让人兴奋，所以可以在两个方法里添加`console.log()`代码来标识方法的执行过程，添加jQuery库和jquery.jqia.contextMenu.js文件到页面中，然后编写如下代码：

```
$('p').init();
```

不幸的是，运行网页，在`console.log()`语句后，居然出错了。原因是使用了公共的方法名，jQuery库已经使用了这个名字。方法名与jQuery定义的`init()`方法冲突，这个问题引入了一个新的知识点：命名空间。

恰当的命名空间插件是开发工作非常重要的一部分。它可以确保插件尽可能不与其他插件冲突，甚至是jQuery的核心库方法。

要使用这条指南原则，可以重新命名方法为`jqiaCustomMenuInit()`和

`jqiaCustomMenuDestroy()`。修改以后,就不会与任何jQuery核心库的方法名冲突了。虽然这个修改可以正常工作,但是jQuery开发指南并不鼓励定义多个命名空间来弄乱`$.fn`。建议的解决方案是,把所有调用的方法集中到一个对象里(通常叫`methods`),然后使用一个包含字符串参数的方法来调用它们,这个方法的参数就是方法名。

为了便于大家理解,假设调用所有方法的方法名为`jqiaContextMenu`。使用它,可以像如下代码一样来执行`destroy()`方法(现在不考虑参数):

```
$('#element').jqiaContextMenu('destroy');
```

遵守这条原则,可以把列表12.1的代码修改如下。

列表 12.2 重构 Jqia Context Menu 插件代码

331

```
(function($) {  
    var defaults = {  
        idMenu: null,  
        bindLeftClick: false  
    };  
    var methods = {  
        init: function(options) {  
            // 这里编写代码...  
        },  
        destroy: function() {  
            // 这里编写代码...  
        }  
    };  
  
    $.fn.jqiaContextMenu = function(method) {  
        if (methods[method]) {  
            return methods[method].apply(  
                this,  
                Array.prototype.slice.call(arguments, 1)  
            );  
        } else if ($.type(method) === 'object') {  
            return methods.init.apply(this, arguments);  
        } else {  
            $.error('Method ' + method +  
                ' does not exist on jQuery.jqiaContextMenu');  
        }  
    };  
})(jQuery);
```

① 定义默认选项

定义包含方法的对象

② 定义 jqiaContextMenu 命名空间并赋值一个匿名函数

如果必备的方法存在,就传递其他参数调用该方法

如果参数是对象,就调用 init()方法

如果都不是,则抛出异常

在最外层的匿名函数里,设置了默认的插件参数值①。然后,声明了一个包含需要方法的对象(空对象体)②。

代码的第二部分最有意思,而且非常灵活。首先,复制匿名函数给`$.fn`的新属性,属性名为`jqiaContextMenu`③。这样做,是为所有方法声明一个名字,而不是每个方法一个名字,正如指南所描述的。在函数内部,我们测试了传入的第一个参数`method`,在`methods`变量内部④。因此,使用了JavaScript高级编程中的`apply()`和`call()`来执行需要的方法。`apply()`用来设置函数上下文(`this`)到jQuery的集合元素,然后把参数转发给调用的方法,除了第一个插

件的参数（因为第一个参数是调用方法的名字）。因为arguments不是真正的数组（类数组对象），所以使用数组的slice()和call()来删除arguments中的第一个元素。

注意：如果要重新复习 apply()和 call()的知识，请阅读附录。

如果第一个测试失败，则检查第一个参数是否是对象❶。此时会调用init()方法，并转发给它所有的jqiaContextMenu()参数。原因是，如果用户调用jqiaContextMenu()，则传递包含选项参数的对象，假设用户想初始化插件并调用默认的方法。此时，也可以使用apply()方法来设置函数上下文为当前jQuery集合元素，然后转发参数给调用的方法。最后，如果两个方法都失败，就可以使用\$.error()工具函数抛出异常❷。

332

正如我们看到的，这个代码更新非常有效，允许为所有方法使用单一命名空间(jqiaContextMenu)。本节讨论的原则适合插件中的事件绑定和数据存储。下面继续学习。

12.3.5 命名空间事件和数据

第6章介绍了命名空间事件的可能性，这在编写插件的时候也非常有用。为插件的命名空间事件添加处理器是最佳实践。遵守这条原则，如果后面需要解绑事件处理器，那么不需要担心删除其他插件中相同的事件处理器。

除了侦听事件，一些插件还需要在页面元素上存储数据。这对于跟踪元素或者检查插件是否在该元素上被调用十分有用。可以使用jQuery的data()方法来实现，这已在第4章介绍过。便于查询和删除。

通过遵循目前为止介绍的所有原则，可以开发出非常棒的jQuery插件。但是还缺少最重要的一块拼图：方法体。下面从init()开始介绍。

Jqia Context Menu 的 init()方法

init()方法有以下职责。

- (1) 检查传递给插件的选项参数，特别是强制参数。
- (2) 使用默认值合并传递的选项参数。
- (3) 测试插件是否在选择元素上初始化。
- (4) 在jQuery集合元素中存储参数选项。
- (5) 侦听鼠标右键点击事件，命名为contextmenu，在jQuery元素中展示自定义菜单。选择侦听鼠标左键点击事件(click)。
- (6) 当在元素外部触发click事件时，隐藏自定义菜单。

要完成第一步，首先要验证idMenu，包含要显示菜单的元素在页面上存在。

下面的代码实现了检查过程：

```
if (!options.idMenu) {  
    $.error('No menu specified');  
} else if ($('#' + options.idMenu).length === 0) {  
    $.error('The menu specified does not exist');  
}
```

在以上代码里，使用length属性来测试页面中是否存在元素。

实现第二步很简单。需要使用jQuery的extend()工具函数来合并值：

```
options = $.extend(true, {}, defaults, options);
```

正如代码中看到的，要重用options来避免添加额外的变量。

第三步和第四步紧密关联。一旦插件在选择元素上初始化完毕，就使用jQuery的data()方法来使用相同的名字存储选项参数。此时，也可以使用它们来检验元素是否被插件初始化。可以使用带有要添加功能的存储信息，比如修改某个元素的配置。

要存储数据，可以编写下面的代码：

```
this.data('jqiaContextMenu', options);
```

当插件第一次在页面中执行时，要确保没有元素被初始化。如果在相同的集合元素上运行Jqia Context Menu插件呢？在相同的元素上两次初始化是我们要避免的情况，因为它会两次添加相同的事件处理器。要检查集合中每个匹配的元素不会使用插件命名空间(jqiaContextMenu)存储数据。这个功能可以通过如下代码实现：

```
if (  
    this.filter(function() {  
        return $(this).data('jqiaContextMenu');  
    }).length !== 0  
) {  
    $.error('The plugin has already been initialized');  
}
```

虽然简短，但代码块给了我们机会来强调重点：插件中this的含义。在附加给\$.fn的函数内，this关键字引用的是jQuery实例（插件调用的jQuery集合）。可以直接使用每个jQuery方法而无须使用\$()包装它（例如\$(this)）。对于methods中定义的函数也一样，因为已经使用apply()修改了它们的上下文。如果没有使用apply()，在init()方法里，this关键字表示引用了methods对象。

基于如何构建的插件，在插件中执行的回调函数里，this引用的是某个具体的DOM元素。在代码里使用了jQuery的filter()方法，并在集合的元素中进行迭代。在第一次迭代里，回调函数的this引用的是匹配集合里的第一个元素；在第二次迭代里，this引用的是集合中的第二个元素，以此类推。这也是为什么在传递给filter()的匿名函数里，传递的this作为\$()

的参数：使用jQuery的data()方法。

既然已经更好地理解jQuery插件中this的意义，下面继续讨论init()方法。

第五步是项目的核心部分。要实现它，需要为contextmenu事件添加一个回调函数，并在用户PC上点击鼠标右键的时候触发。应该还记得前面提供的点击鼠标左键的处理器。基于开发者传递的选项参数，或许需要同时侦听contextmenu和click。

在回调函数里需要阻止默认的行为；否则，原始的上下文菜单将会显示出来。阻止默认行为后，就必须根据鼠标点击的位置来设置自定义菜单的位置（信息存储在传递给回调函数的Event对象里）。最后，必须显示菜单。

执行这个操作的代码如下：

```
this.on(
    'contextmenu.jqiaContextMenu' +
    (options.bindLeftClick ? ' click.jqiaContextMenu' : ''),
    function(event) {
        event.preventDefault();

        $('#' + options.idMenu)
            .css({
                top: event.pageY,
                left: event.pageX
            })
            .show();
    }
);
```

以上代码使用三元运算符来判断是否为click事件建立了处理器。传递一个对象给jQuery的css()方法来设置菜单的位置（在菜单上需要设置position: absolute，在CSS文件里定义）。此外没有设置单位（像素），因为当不指定时，jQuery假设值的单位是像素。

最后一步，当点击时，无论是点击鼠标的左右键，还是隐藏自定义菜单，在元素外部通过Jqia Context Menu初始化。这意味着应该附加一个处理器，除了插件初始化的元素，页面所有元素都应该隐藏自定义菜单。为页面所有元素添加处理器有严重的性能问题，所以会使用事件委托来实现。只为文档的根元素添加一个侦听器，html元素，代码如下：

```
$('#html').on(
    'contextmenu.jqiaContextMenu click.jqiaContextMenu',
    function() {
        $('#' + options.idMenu).hide();
    }
);
```

◀ 335

编写完这些代码，或许看起来已经完成了init()方法，其实还没有结束。

当前状态下，项目有个严重的bug。当点击初始化某个元素时，会显示一个自定义菜单。然后，因为事件冒泡机制，事件会往DOM树根部传播。一旦到达html元素，就会执行添加的回调函数，隐藏自定义菜单。结果是自定义菜单只显示几毫秒（太快了，根本看不清）。要修复这个bug，还必须在初始化元素代码里调用event.stopPropagation()方法。

有了这些代码，`init()`方法就完成了。现在看看如何开发`destroy()`方法（如果好奇完成的插件是什么样子，则可以直接跳到列表12.3）。

Jqia Context Menu 的 `destroy()`方法

`destroy()`方法负责清理插件使用的资源，它包括初始化元素存储的数据和附加的侦听器，还包含附加给html元素的。除此之外，还要确保菜单在`destroy()`方法完成之前隐藏起来；否则它会一直显示在页面上，直到下次加载页面。

满足这个需求的代码实现版本如下所示：

```
this
  .each(function() {
    var options = $(this).data('jqiaContextMenu');
    if (options !== undefined) {
      $('#'+ options.idMenu).hide();
    }
  })
  .removeData('jqiaContextMenu')
  .add('html')
  .off('.jqiaContextMenu');
```

首要的工作就是迭代匹配集合中的每个元素，检查自定义菜单，再隐藏它（如果插件初始化了该元素）。然后删除每个元素上存储的数据，但是这次不需要迭代每个元素并进行检查，所以可以使用jQuery的`removeData()`方法。

最后一件要做的事情就是解绑所有附加给jqiaContextMenu命名空间的事件处理器。为此，要添加html元素到当前的jQuery集合，然后调用`off()`方法，传递".jqiaContextMenu"给它。

最细心的读者可能已经注意到，虽然目的不同，但前面的代码多次使用到"jqiaContextMenu"字符串。为了避免重复声明，可以把它存储到某个私有变量里，然后根据需要修改代码。

336

假设下面的代码已经添加到defaults变量后，

```
var namespace = 'jqiaContextMenu';
```

`destroy()`的方法体可以重写为：

```
this
  .each(function() {
    var options = $(this).data(namespace);
    if (options !== undefined) {
      $('#'+ options.idMenu).hide();
    }
  })
  .removeData(namespace)
  .add('html')
  .off('.' + namespace);
```

此时，插件可以工作，但是还有两个额外的重要知识点需要学习。

12.3.6 维护链式调用性

本书已经多次使用jQuery的链式调用的单行代码执行多个操作。若方法不返回值，就会返回undefined。因为使用插件的开发者在调用jqiaContextMenu()后就无法再调用别的方法了。

插件中维护链式调用性的代码简单且有价值。我们要做的就是确保方法一直返回this关键字。在如下这行代码后修改destroy()方法：

```
.off('.', ' + namespace);
```

添加如下代码：

```
return this;
```

也会为插件的init()方法添加相同的修改。这样修改以后就维护了链式调用性，允许其他使用Jqia Context Menu插件的开发者在单个语句里继续操作相同的集合元素。

12.3.7 允许公开访问默认设置

插件还不够定制化，且当事情变得更复杂后，我们发现要给不同的元素一遍一遍地传递大量相同的参数选项。

一种改进的办法是，为使用插件的开发者暴露默认的设置，这样他们就可以修改设置了。有了这个修改，开发者只需要传递一个对象给默认的插件即可。

需要对项目做两个修改。第一个修改针对defaults变量。为了暴露给外部世界，需要把它赋值给\$.fn属性。要避免违背声明多个命名空间的原则，就需要设置包含默认值的对象作为jqiaContextMenu的属性。因此，代码修改如下：

```
var defaults = {  
    //这里编写参数选项...  
};  
into  
$.fn.jqiaContextMenu.defaults = {  
    //这里编写参数选项...  
};
```

需要把默认的配置移到声明命名空间(\$.fn.jqiaContextMenu=function (method) {})的代码后面。

有了这个更新，使用defaults变量就无法再引用任何默认值了。必须使用\$.fn.jqiaContextMenu.defaults替换代码中的每个defaults变量。当使用默认值来合并参数选项时，在我们的项目里它只出现了一次，即驻留在init()方法里。因此必须修改代码语句，如下：

```
options = $.extend(true, {}, $.fn.jqiaContextMenu.defaults, options);
```

修改完毕，下面看看如何使用暴露的默认值。

假设当调用Jqia Context Menu时想绑定左键点击事件，那么可以这样编写代码：

```
$.fn.jqiaContextMenu.defaults.bindLeftClick = true;
```

然后只传递对象里的idMenu属性来调用插件。

最后一次修改后，就完成了整个项目。最后的代码如列表12.3所示。

列表 12.3 Jqia Context Menu 的最终版本

```
(function($) {  
    var namespace = 'jqiaContextMenu';  
  
    var methods = {  
        init: function(options) {  
            if (!options.idMenu) {  
                $.error('No menu specified');  
            } else if ($('#' + options.idMenu).length === 0) {  
                $.error('The menu specified does not exist');  
            }  
  
            options = $.extend(  
                true,  
                {},  
                $.fn.jqiaContextMenu.defaults,  
                options  
            );  
  
            if (  
                this.filter(function() {  
                    return $(this).data(namespace);  
                }).length !== 0  
            ) {  
                $.error('The plugin has already been initialized');  
            }  
  
            this.data(namespace, options);  
  
            $('html').on(  
                'contextmenu.' + namespace + ' click.' + namespace,  
                function() {  
                    $('#' + options.idMenu).hide();  
                }  
            );  
  
            this.on(  
                'contextmenu.' + namespace +  
                (options.bindLeftClick ? ' click.' + namespace : ''),  
                function(event) {  
                    event.preventDefault();  
                    event.stopPropagation();  
  
                    $('#' + options.idMenu)  
                        .css({
```

```

        top: event.pageY,
        left: event.pageX
    })
    .show();
    }
    );

    return this;
},
destroy: function() {
    this
        .each(function() {
            var options = $(this).data(namespace);
            if (options !== undefined) {
                $('#'+ options.idMenu).hide();
            }
        })
        .removeData(namespace)
        .add('html')
        .off('.' + namespace);

    return this;
}
});

$.fn.jqiaContextMenu = function(method) {
    if (methods[method]) {
        return methods[method].apply(
            this,
            Array.prototype.slice.call(arguments, 1)
        );
    } else if ($.type(method) === 'object') {
        return methods.init.apply(this, arguments);
    } else {
        $.error('Method ' + method +
            ' does not exist on jQuery.jqiaContextMenu'
        );
    }
};

$.fn.jqiaContextMenu.defaults = {
    idMenu: null,
    bindLeftClick: false
};
})(jQuery);

```

339

以上代码可以在本书的js/jquery.jqia.contextMenu.js例子里找到。要想让此插件正常工作，还需要在页面里添加样式文件引用，可以在css/jquery.jqia.contextMenu.css中找到文件。我们也提供了一个demo，可以在chapter-12/jqia.contextMenu.html里找到，这样就可以使用插件了。图12.1展示了在Jqia Context Menu插件初始化的元素上点击鼠标右键的效果。

你认为结果怎么样？开心吗？希望大家享受开发此插件的过程。第12.4节会学习更加复杂的jQuery插件，根据之前学习的插件编程指南原则进行构建。

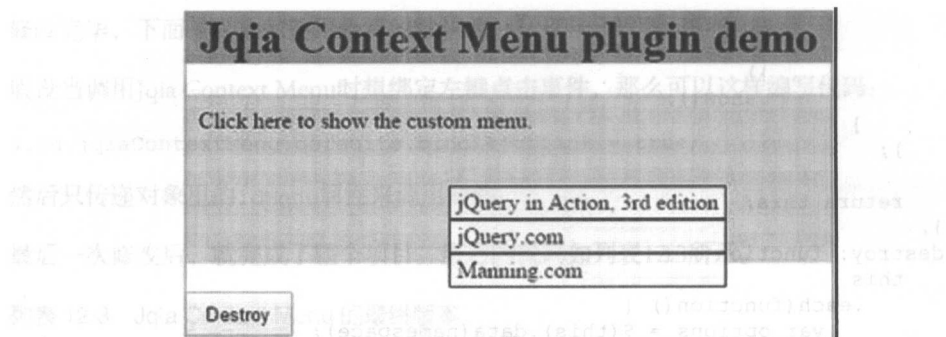


图 12.1 在 Jqia Context Menu 插件初始化的元素上点击鼠标右键的效果

12.4 演示：创建 jQuery 幻灯片插件

Demo: creating a slideshow as a jQuery plugin

对于更加复杂的插件例子，需要开发一种 jQuery 方法，允许开发者快速构建一个幻灯片页面。首先会创建一个 jQuery 插件，它的名字叫 Jqia Photomatic；然后会创建一个测试页面。当完成时，这个测试页面如图 12.2 所示。

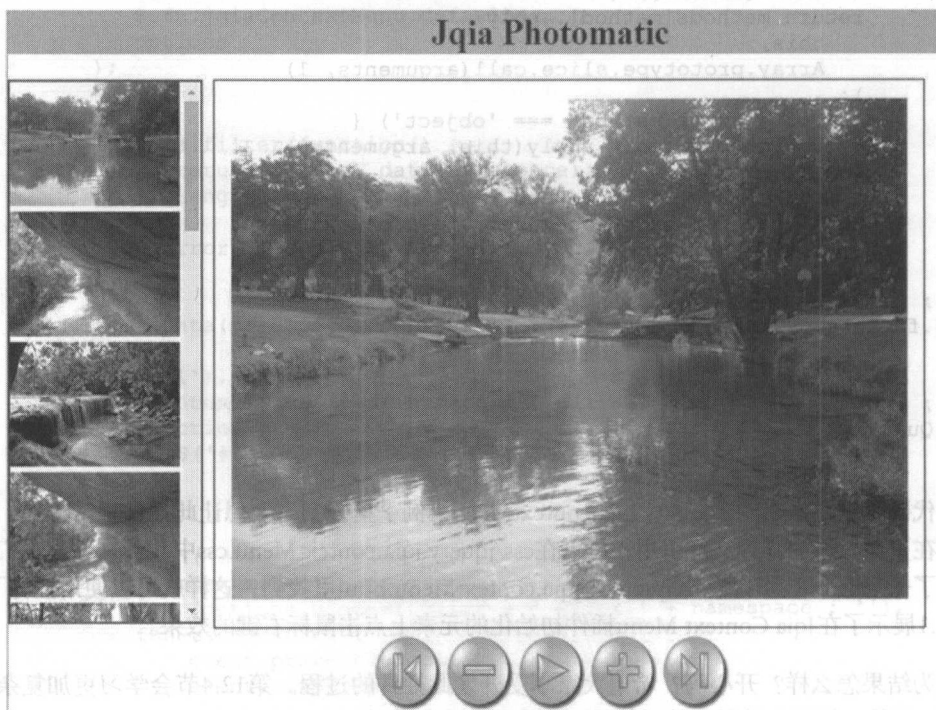


图 12.2 使用 Jqia Photomatic 插件的测试页面效果

这个页面包含下面的组件。

- 一系列缩略图。
- 缩略图列表中的一张全尺寸图片。
- 一系列控制图片显示的按钮。

页面的行为如下：

- 点击任意一张缩略图，显示对应的全尺寸图片。
- 点击全尺寸图片显示下一张图片。
- 点击任意按钮执行下面的操作：
 - First——显示第一张图片。
 - Previous——显示上一张图片。
 - Next——显示下一张图片。
 - Last——显示最后一张图片。
 - Play——自动显示图片直到点击按钮。
- 图像列表末尾的任何向后操作都会从头开始，而列表开头的任何向前操作都会跳转至末尾继续。点击最后一张图片的“Next”按钮会显示第一张图片，点击第一张图片的“Previous”按钮会显示最后一张图片。

340

我们会定义插件，这样开发者可以以任何方式设置他们自己喜欢的元素，并告诉我们使用哪个页面元素的目的。此外，为了给Web开发者更大的自由空间，我们会定义插件，只要这些图片集中在测试的页面中，开发者就可以提供包含图片缩略图的jQuery集合。

下面是Jqia Photomatic插件的语法。

341

jqiaPhotomatic()方法语法

jqiaPhotomatic(options)

构造缩略图集合，还有options对象里标识的页面元素，作为Jqia Photomatic插件的控件使用。

参数

options (Object) 为Jqia Photomatic插件定义参数选项。参考表12.1的详细介绍。

返回

jQuery集合。

因为有许多参数控制Jqia Photomatic插件的操作(有些参数可以忽略)，所以可以使用第12.3.3节讨论的哈希对象来传输参数。可能的选项参数如表12.1所示。

表 12.1 Jqia Photomatic 自定义插件方法的参数选项

名 称	描 述
firstControl	(Selector)可以标识 DOM 元素的作为第一个控件的 jQuery 选择器。如果忽略，则不会设置控件
lastControl	(Selector)可以标识 DOM 元素的作为最后一个控件的 jQuery 选择器。如果忽略，则不会设置控件
nextControl	(Selector)可以标识 DOM 元素的作为下一个控件的 jQuery 选择器。如果忽略，则不会设置控件
photoElement	(Selector)区分全尺寸图片的 jQuery 选择器。如果忽略，则默认是 jQuery 选择器 img.photomatic-photo
playControl	(Selector)可以标识 DOM 元素的作为播放控件的 jQuery 选择器。如果忽略，则不会设置控件
previousControl	(Selector)可以标识 DOM 元素的作为上一个控件的 jQuery 选择器。如果忽略，则不会设置控件
transformer	(Function)用来把缩略图 URL 转换成对应的全尺寸图片 URL 的函数。如果忽略，则默认的转换信息会使用 URL 里的 photo 替换所有的 thumbnail 实例
delay	(Number)自动幻灯片切换之间的时间间隔，以毫秒计算。默认是 3000 毫秒

342 现在要在创建 Jqia Photomatic 插件之前先创建测试页面。

12.4.1 设置标签

第一步要先创建使用插件的页面。代码可以在chapter-12/jqia.photomatic.html中找到，如列表 12.4 所示。

列表 12.4 图 12.2 中显示的创建 Photomatic 的测试页面

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Jqia Photomatic - jQuery in Action, 3rd edition</title>
    <link rel="stylesheet" href="../css/main.css"/>
    <link rel="stylesheet" href="../css/jquery.jqia.photomatic.css"/>
  </head>
  <body>
    <h1 class="header">Jqia Photomatic</h1>
    <div id="thumbnails-pane">
      
      
      
      
      
      
      
      
    </div>
  </body>
</html>
```

包含缩略图

```





</div>

```

② 为全尺寸照片定义图片元素

```

<div>
  <img id="photo-display" src="" title="Click for next photo" />
</div>

```

包含作为控
件的元素 ③

```

<div id="button-bar">
  
  
  
  
  
</div>

```

```

<script src="../../js/jquery-1.11.3.min.js"></script>
<script src="../../js/jquery.jqia.photomatic.js"></script>
<script>

```

```

  $('#thumbnails-pane img').jqiaPhotomatic({
    photoElement: '#photo-display',
    previousControl: '#previous-button',
    nextControl: '#next-button',
    firstControl: '#first-button',
    lastControl: '#last-button',
    playControl: '#play-button',
    delay: 1000
  });
</script>

```

调用 Photomatic
插件 ④

343

通过应用“低调的JavaScript”原则，可以把所有样式文件都存放到外部CSS文件里，代码就变得简洁很多。

HTML标签由一个包含缩略图的容器组成①。一张图片元素显示全尺寸图片②，一个集合元素③会控制图片的显示。其他所有事情都由插件处理。页面的脚本很简单，由调用插件的一行代码组成，传递了一些选项参数④。

编写完这些标签代码后，现在是时候深入介绍插件了。

12.4.2 开发 Jqia Photomatic 插件

要开发插件，则要使用与Jqia Context Menu插件一样的骨架，并采用不同的命名空间：

```

(function($){
    var methods = {
        init: function() {
        }
    };

    $.fn.jqiaPhotomatic = function(method) {
        if (methods[method]) {
            return methods[method].apply(
                this,
                Array.prototype.slice.call(arguments, 1)
            );
        } else if ($.type(method) === 'object') {
            return methods.init.apply(this, arguments);
        } else {
            $.error('Method ' + method +
                ' does not exist on jQuery.jqiaPhotomatic'
            );
        }
    };
})(jQuery);

```

344 插件只需要初始化函数，但是，为了支持后期扩展，可以使用`methods`变量，与前面的项目一样。

在`init()`函数内，使用表12.1介绍的默认值来合并调用者设置。结果存储在单个`options`对象里，可以在后面的函数里访问到。

插件调用者可能想重写默认值（例如`delay`属性），所以会为外部暴露接口，代码如下：

```

$.fn.jqiaPhotomatic.defaults = {
    photoElement: 'img.photomatic-photo',
    transformer: function(name) {
        return name.replace('thumbnail', 'photo');
    },
    nextControl: null,
    previousControl: null,
    firstControl: null,
    lastControl: null,
    playControl: null,
    delay: 3000
};

```

与Jqia Context Menu插件的做法类似，合并是通过jQuery的`$.extend()`方法进行的：

```
options = $.extend(true, {}, $.fn.jqiaPhotomatic.defaults, options);
```

执行这行代码后，`options`会包含调用者重写的值。

还要保证追踪一些其他东西。为了插件能正常工作，比如，下一张相关的图片，上一张相关的图片，不仅要显示缩略图列表，还要标识当前显示的图片。

缩略图列表就是jQuery集合，也就是此方法操作的对象——或者至少应该是。我们不知道开发者在jQuery集合里收集什么对象，所以需要过滤图片元素。这个操作可以使用选择器和

jQuery的filter()方法实现。但是在哪里存储这些信息呢？

可以轻易创建另外一个变量来存储，但是，为了确保合并设置信息，应该把它作为options的一个属性。为此，必须细心修改extend()方法，代码如下：

```
options = $.extend(
    true,
    {},
    $.fn.jqiaPhotomatic.defaults,
    options,
    {
        current: 0,
        $thumbnails: this.filter('img')
    }
);
```

◀ 345

应注意如何存放当前显示的图片，以及缩略图列表作为extend()方法的最后一个参数，因为后者决定了属性合并的优先级。我们把列表属性命名为\$thumbnails，因为它的值是jQuery集合。

既然初始化状态设置完毕，现在就应该继续编写插件的核心代码了——添加适当的功能来显示控件、缩略图和图片。

注意：能够保存状态，是因为有闭包（closure）。虽然前面已经介绍过闭包，但是，如果还不理解，请阅读本书的附录部分。必须理解闭包的概念，不仅是为了完整实现 Jqia Photomatic 插件，还为了以后能编写更复杂的插件功能代码。

现在需要给控件和元素添加许多事件侦听器。因为当声明表示侦听器的函数时，options变量在范围内，每个侦听器都会作为包含options变量闭包的一部分。可以确保，虽然后者只是临时出现，它表示的状态仍然能够被定义的所有侦听器使用。

说到这些侦听器，这里是click事件侦听器的列表，还需要附加到不同的元素上。

- 点击缩略图显示全尺寸图片。
- 点击全尺寸图片，会显示下一张图片。
- 点击“Previous”按钮会显示上一张图片。如果当前显示第一张图片，就显示最后一张图片。
- 点击“Next”按钮会显示下一张图片。如果当前显示最后一张图片，就显示第一张图片。
- 点击“First”按钮显示第一张图片。
- 点击“Last”按钮显示最后一张图片。
- 点击“Play”按钮，会自动按照顺序播放图片。再点击会停止播放。

仔细看这个列表，我们发现这些侦听器有些相同的地方：都需要全尺寸显示缩略图。作为一个聪明的程序员，可以分解出共同的代码到一个函数里，这样就不需要每次重复编写这种代码。

346 并不想把这种插件内部使用的代码放到全局命名空间或者\$命名空间。JavaScript函数语言的强大威力又显示出来，允许在插件函数里定义这个新函数。这样，就限制了它的使用范围在插件本身（我们的目标之一）。

出于这个原因，我们定义了这个需要的函数，命名为showPhoto()，在插件内部，但是在init()外部。这个函数定义了两个参数：第一个是实际的插件调用的参数选项，第二个是要展示的全尺寸图片的索引编号。函数的代码如下：

```
function showPhoto(options, index) {  
    $(options.photoElement).attr(  
        'src',  
        options.transformer(options.$thumbnails[index].src)  
    );  
    options.current = index;  
}
```

新的函数，当传递全尺寸显示图片的索引后，使用options对象中的值来执行下面的操作：

(1) 查找索引标识图片的src特性。

(2) 传值给transformer函数，以把缩略图URL转换为全尺寸图片URL。

(3) 把转换的值赋给全尺寸图片元素的src属性。

(4) 记录新的索引为当前显示的图片索引。

在这个函数编写完毕后，就可以定义这些侦听器了。我们会从为缩略图添加函数开始，只需要显示它们对应的全尺寸图片即可，代码如下：

```
options.$thumbnails.click(function() {  
    showPhoto(options, options.$thumbnails.index(this));  
});
```

在处理器内，通过传递点击元素（通过this引用）给jQuery的index()方法获取缩略图的索引值。

让图片显示下一个列表图片也非常简单：

```
$(options.photoElement + ', ' + options.nextControl).click(function() {  
    showPhoto(options, (options.current + 1) % options.$thumbnails.length);  
});
```

我们建立了一个点击事件处理器，在其中调用了showPhoto()函数，使用了options对象和一个索引值。注意如何使用JavaScript的取模运算符(%)来处理点击最后一张图片的显示问题。

细心的读者会发现实际上包括在声明中的另外一个选择器，原因是Next按钮的行为几乎相同。

347 可以通过逗号优化语句指令创建单个选择器。

First、Previous、Last按钮的处理器都遵循类似的模式：找出要显示的缩略图的索引，然后使用此索引值调用showPhoto()，代码如下：

```

$(options.previousControl).click(function() {
    showPhoto(
        options,
        options.current === 0 ?
        options.$thumbnails.length - 1 :
        options.current - 1
    );
});

$(options.firstControl).click(function() {
    showPhoto(options, 0);
}).triggerHandler('click');

$(options.lastControl).click(function() {
    showPhoto(options, options.$thumbnails.length - 1);
});

```

代码里唯一一行值得注意的代码是`triggerHandler()`。调用此方法来加载图片到插件执行的图片容器里。

播放按钮的实现复杂得多。这个控件必须播放所有的图片而不是显示某个特定的图片。当用户再次点击的时候必须停止轮播幻灯片效果。下面看看具体的实现代码：

```

var tick;
$(options.playControl).click(function() {
    var $this = $(this);
    if ($this.attr('src').indexOf('play') !== -1) {
        tick = window.setInterval(
            function() {
                $(options.nextControl).triggerHandler('click');
            },
            options.delay
        );
        $this.attr(
            'src',
            $this.attr('src').replace('play', 'pause')
        );
    } else {
        window.clearInterval(tick);
        $this.attr(
            'src',
            $this.attr('src').replace('pause', 'play')
        );
    }
});

```

首先使用图片的`src`来决定执行何种操作。如果`src`属性包含字符串“play”，则需要开始幻灯片播放；否则就停止它。

◀ 348

在`if`体内，使用JavaScript的`setInterval()`方法来触发函数的连续执行，使用`delay`值延迟操作。存储时间间隔器的处理器在变量`tick`中。在传递给`setInterval()`的匿名函数内部，使用Next控件的点击操作显示下一张图片；这会在每次调用`setInterval()`的内部函数里执行一次。

if体内最后的代码更新要显示停止图片的src属性。这个修改允许第二次点击“Play”按钮时执行else体代码。

在else代码部分里，要停止轮播。为此，使用clearInterval()清除间隔超时，传递tick，并且恢复显示图片。

最后的任务就是，为了维护链式调用性原则，需要返回匹配元素的最初集合。可以使用以下代码实现：

```
return this;
```

喝点小酒庆祝一下；我们成功喽！可以打赌，大家都认为这个过程没有如此简单。完成的插件代码可以在本书的js/jquery.jqia.photomatic.js中找到，列表12.5的代码显示如下。

列表 12.5 插件的完整实现代码

```
(function($){  
    function showPhoto(options, index) {  
        $(options.photoElement).attr(  
            'src',  
            options.transformer(options.$thumbnails[index].src)  
        );  
        options.current = index;  
    }  
  
    var methods = {  
        init: function(options) {  
            options = $.extend(  
                true,  
                {},  
                $.fn.jqiaPhotomatic.defaults,  
                options,  
                {  
                    current: 0,  
                    $thumbnails: this.filter('img')  
                }  
            );  
  
            options.$thumbnails.click(function() {  
                showPhoto(options, options.$thumbnails.index(this));  
            });  
  
            $(options.photoElement + ', ' + options.nextControl).click(  
                function() {  
                    showPhoto(  
                        options,  
                        (options.current + 1) % options.$thumbnails.length  
                    );  
                }  
            );  
  
            $(options.previousControl).click(function() {  
                showPhoto(  
                    options,  
                    (options.current - 1 + options.$thumbnails.length) % options.$thumbnails.length  
                );  
            });  
        }  
    };  
  
    $.fn.jqiaPhotomatic = function(options) {  
        return this.each(function() {  
            methods.init.call(this, options);  
        });  
    };  
});
```

```

options.current === 0 ?
    options.$thumbnails.length - 1 :
    options.current - 1
    );
});

$(options.firstControl).click(function() {
    showPhoto(options, 0);
}).triggerHandler('click');

$(options.lastControl).click(function() {
    showPhoto(options, options.$thumbnails.length - 1);
});

var tick;
$(options.playControl).click(function() {
    var $this = $(this);
    if ($this.attr('src').indexOf('play') !== -1) {
        tick = window.setInterval(
            function() {
                $(options.nextControl).triggerHandler('click');
            },
            options.delay
        );
        $this.attr(
            'src',
            $this.attr('src').replace('play', 'pause')
        );
    } else {
        window.clearInterval(tick);
        $this.attr(
            'src',
            $this.attr('src').replace('pause', 'play')
        );
    }
});

return this;
}

};

$.fn.jqiaPhotomatic = function(method) {
    if (methods[method]) {
        return methods[method].apply(
            this,
            Array.prototype.slice.call(arguments, 1)
        );
    } else if ($.type(method) === 'object') {
        return methods.init.apply(this, arguments);
    } else {
        $.error(
            'Method ' + method +
            ' does not exist on jQuery.jqiaPhotomatic'
        );
    }
};

```

```
$.fn.jqiaPhotomatic.defaults = {
  photoElement: 'img.photomatic-photo',
  transformer: function(name) {
    return name.replace('thumbnail', 'photo');
  },
  nextControl: null,
  previousControl: null,
  firstControl: null,
  lastControl: null,
  playControl: null,
  delay: 3000
};
})(jQuery);
```

以上代码是典型的启用jQuery的代码；它把强大的功能包装在紧凑的代码里。但是它却展示了一系列重要技巧——使用闭包来维护跨jQuery插件域的状态，而且允许插件创建内部实现函数，不需要修改任何命名空间。

还要注意的，因为特别留意到不会向外部泄露状态信息，所以可以在同一个页面多次使用Jqia Photomatic插件，而不需要担心它们之间互相冲突（当然要确保页面里没有重复的ID）。



但是插件已经完成了吗？下面的问题留给大家作为思考和练习。

- 图片之间的转换很快。使用动画与特效一章学习的知识，修改插件代码，这样图片就可以交叉淡入淡出。
- 继续深入，如何允许开发者自定义动画效果？
- 为了最大的灵活性，使用用户创建的HTML元素编写这个插件。如何开发一个类似的插件，但是少点灵活性，自动生成所有需要的HTML元素？

现在既然已经知道了如何实现一个新的jQuery方法，是时候学习如何创建自定义工具函数了。

12.5 编写自定义工具函数

Writing custom utility functions

本书使用工具函数（utility function）一词来描述作为jQuery（简称\$）的属性函数。这些函数通常在非JavaScript对象上执行操作，或者执行一些其他非对象操作。一些工具函数的例子是\$.each()和\$.noConflict()。本节将会学习如何添加自定义工具函数。

添加函数作为对象或者函数的属性，都是非常简单的，像声明函数并赋值给它一样。创建自定义工具函数也像下面的代码这样简单：

```
$.say = function(what) {
  alert('I say ' + what);
};
```

而且事实就是这样简单。但是这样定义工具函数并非没有副作用。如果某些开发者在页面中

使用Prototype框架，并且调用\$.noConflict()，代码就会在Prototype的\$()函数中而不是jQuery函数上进行扩展（如果概念不懂，可以阅读附录的方法一节内容）。正如我们看到的，与操作匹配集合元素的插件不同，工具函数是复制给\$或者\$.fn。

已经在第12.3.2节里介绍过问题和解决版本（提示：创建一个IIFE）。讨论刚才这样的工具函数不是什么大问题，先来实现一个简单的例子。

12.5.1 编写日期格式化器

如果进行客户端编程，其中一项重要的事情就是日期的格式化，这是JavaScript的Date对象没有提供的功能。因为这种函数会操作在Date对象而不是具体的DOM元素上，所以好的解决方法就是定义为工具函数。使用下面的语法来编写此工具函数。

\$.formatDate()函数语法

\$.formatDate(date, pattern)

根据提供的模式来格式化传入的日期。模式中取代的字符如下：

yyyy: 四位数的年

yy: 两位数的年

MMMM: 月的完整名称

MMM: 月的简称

MM: 两位数的月

M: 月的数字

dd: 两位数的天

d: 一个月的天数

EEEE: 一周某一天的完整名称

EEE: 一周某一天的简称

a: 经略（AM 或者 PM）

HH: 24时制的两位数的小时

H: 24时制的小时

hh: 12时制的两位数的小时

h: 12时制的小时

mm: 两位数的分钟

m: 小时中的分

ss: 两位数的秒

s: 分钟的秒

S: 三位数的毫秒

参数

date(Date) 要格式化的日期

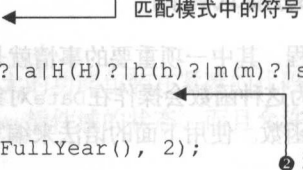
pattern(String) 格式化日期的模式。任意没有匹配的字符会直接拷贝到结果中。

返回

格式化的日期。

列表12.6展示了这个函数的实现代码。不会过度介绍日期格式化的算法，因为这不是本书的主要关注点。在创建复杂函数的时候,会讨论一些有用的策略。

列表 12.6 \$.formatDate()工具函数的实现



```
(function($) {
    var patternParts =
    /^((yy(yy)?|M(M(M(M)?)?|d(d)?|EEE(E)?|a|H(H)?|h(h)?|m(m)?|s(s)?|S)/);
    var patternValue = {
        yy: function(date) {
            return toFixedWidth(date.getFullYear(), 2);
        },
        yyyy: function(date) {
            return date.getFullYear().toString();
        },
        MMMM: function(date) {
            return $.formatDate.monthNames[date.getMonth()];
        },
        MMM: function(date) {
            return $.formatDate.monthNames[date.getMonth()].substr(0, 3);
        },
        MM: function(date) {
            return toFixedWidth(date.getMonth() + 1, 2);
        },
        M: function(date) {
            return date.getMonth() + 1;
        },
        dd: function(date) {
            return toFixedWidth(date.getDate(), 2);
        },
        d: function(date) {
            return date.getDate();
        },
        EEEE: function(date) {
            return $.formatDate.dayNames[date.getDay()];
        },
        EEE: function(date) {
            return $.formatDate.dayNames[date.getDay()].substr(0, 3);
        },
        HH: function(date) {
            return toFixedWidth(date.getHours(), 2);
        },
        H: function(date) {
            return date.getHours();
        },
        hh: function(date) {
            var hours = date.getHours();
            return toFixedWidth(hours > 12 ? hours - 12 : hours, 2);
        },
        h: function(date) {
            var hours = date.getHours();
            return toFixedWidth(hours > 12 ? hours - 12 : hours, 2);
        }
    };
    $.formatDate = function(date, pattern) {
        return patternValue[pattern];
    };
})(jQuery);
```

① 定义正则表达式来匹配模式中的符号

② 定义一个包含格式化函数的读/写，一旦找到匹配的元素就执行

```

    return date.getHours() % 12;
  },
  mm: function(date) {
    return toFixedWidth(date.getMinutes(), 2);
  },
  m: function(date) {
    return date.getMinutes();
  },
  ss: function(date) {
    return toFixedWidth(date.getSeconds(), 2);
  },
  s: function(date) {
    return date.getSeconds();
  },
  S: function(date) {
    return toFixedWidth(date.getMilliseconds(), 3);
  },
  a: function(date) {
    return date.getHours() < 12 ? 'AM' : 'PM';
  }
};

function toFixedWidth(value, length, fill) {
  ④ 格式化传入值为固定的宽度
  var result = (value || '').toString();
  ① 设置默认值
  fill = fill || '0';
  var padding = length - result.length;
  ⑤ 计算内边距
  if (padding < 0) {
    result = result.substr(-padding);
    ⑥ 需要时截取
  } else {
    for (var n = 0; n < padding; n++) {
      result = fill + result;
      ⑦ 排版结果
    }
    ⑧ 返回最终结果
    return result;
  }
}

$.formatDate = function(date, pattern) {
  ⑨ 实现函数主体
  var result = [];
  while (pattern.length > 0) {
    patternParts.lastIndex = 0;
    var matched = patternParts.exec(pattern);
    if (matched) {
      result.push(patternValue[matched[0]].call(this, date));
      pattern = pattern.slice(matched[0].length);
    } else {
      result.push(pattern.charAt(0));
      pattern = pattern.slice(1);
    }
  }
  return result.join('');
};

$.formatDate.monthNames = [
  ⑩ 提供月份名
  'January', 'February', 'March', 'April', 'May', 'June', 'July',
  'August', 'September', 'October', 'November', 'December'
];

⑪ 提供日期名
$.formatDate.dayNames = [

```

```

        'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
        'Saturday'
    ];
})(jQuery);

```

实现代码最有意思的地方就是，除了一些用来检查日期的JavaScript技巧，就是赋值给\$.formatDate的匿名函数⑨需要一些辅助数据、对象和函数来完成工作。特别是：

- 匹配模式中字符的正则表达式①。
- 月的英文名字的列表⑩。
- 日期的英文名字的列表⑪。
- 为每个字符类型提供值的子函数集合，给定一个日期源⑫。
- 为固定宽度的字符使用专门的长度格式化传入的值⑬。

在这个函数里，定义了集合变量和函数。有些是私有的（使用var声明），而有些暴露给外部（定义为\$.formatDate的属性）。patternParts变量是唯一函数所需要的，但是没有必要暴露给外部，保持私有。换句话说，monthNames⑩和dayNames⑪或许被重新提供月和日期给其他语言，允许它们从外部访问。记住，JavaScript函数是一级对象，它们和其他JavaScript对象一样有自己的属性。

355 函数依赖于toFixedWidth()的支持。这个函数会把传值转换为它等价的字符串，而且会使用默认值填充⑭。然后计算需要的内边距⑮。

如果是负边距（结果比传递的字段长度长），则可以使用专门的长度来截断结果⑯；否则，会使用适当的字符来填充结果⑰而不是直接返回函数结果⑱。

格式化算法本身？简言之，它执行如下操作。

- (1) 创建数组用来保存结果。
- (2) 迭代模式，作为函数第二个参数的模式，处理标识的符号和非符号字符，直到字符串被完整处理。
- (3) 通过设置lastIndex为0来为每个迭代重置正则表达式(存储在patternParts中)。
- (4) 根据当前开始的模式，测试一个符号匹配的正则表达式。
- (5) 调用patternValue集合中的函数，如果匹配，则从Date实例中获取适当的值。这个值会存储到结果数组的尾部，匹配的符号会从模式开头部分删除。
- (6) 从模式里删除第一个字符，如果在当前模式的开始部分符号不匹配，则添加它到结果数组。
- (7) 当整个模式被处理完毕时，把结果数组拼接到一个字符串后作为函数的结果返回。

可以在js/jquery.jqia.formatDate.js中找到此工具函数的源代码，且可以在chapter-12/jqia.formatDate.html的简单页面里测试它的功能。

12.6 总结

Summary

本章介绍了如何编写可复用代码来扩展jQuery框架。作为jQuery扩展编写自己的代码有许多好处。无论使用什么jQuery API，不仅可在Web应用程序里保持代码的一致性，还可以利用jQuery已有的强大功能。

遵守一些命名原则可以避免与其他插件的代码和文件冲突，还有就是\$被页面重新分配了。此外，也介绍了如何构建不破坏jQuery链式调用性的插件构建过程。

创建新的工具函数与在\$中创建新函数属性一样简单，而且新的jQuery方法更容易作为\$.fn的属性创建。

◀ 356

如果编写插件代码让大家感到迷惑，那么强烈推荐大家下载已有的插件来看作者是如何实现其功能的。我们会看到本章介绍的技巧是如何广泛应用到插件编写的过程中的，而且会学习新的技术。

既然已经学习了更多的jQuery编程的知识，现在继续学习如何使用jQuery来管理异步函数。

◀ 357

使用Deferred避免回调地狱

Avoiding the callback hell with Deferred

本章内容

- 什么是promise（承诺）以及为什么它很重要。
- Deferred对象。
- 如何管理异步操作。
- 解析和拒绝promise（承诺）。

很长一段时间，JavaScript开发者使用回调函数来执行一些任务，比如一段时间后执行操作（使用`setTimeout()`），或者定时执行操作（使用`setInterval()`），或者响应某个事件（`click`、`keypress`、`load`等）。我们已经讨论并使用回调函数扩展执行异步操作；例如，第6章关注在事件上，第8章讨论了动画特效，第10章讲解了Ajax。回调函数虽然简单，方便完成工作，但是，当需要执行许多异步操作时变得难以管理。对于内嵌很多回调函数的场景，或者必须同步执行的独立回调函数，通常称为“回调地狱（callback hell）”。

今天，网站使用JavaScript代码多余后台代码（这是API驱动的时代，不是吗？）。因此，开发者需要一种更好的方式来管理和同步异步操作，以包装其代码的可读性和可维护性。

本章将会讨论promise（承诺），它们是什么，jQuery如何使用它。可爱的jQuery库实现了promise的概念，使用了两个对象，即Deferred和Promise。jQuery如何实现promise已经成了讨论、批评的主题以及大多数的变化，后面章节会介绍之。

当学习本章时，我们注意到术语可能让人感到困惑，promise（承诺）的概念并不与Promise对象一一对应，这太诡异了。讲解的过程中穿插了一些例子代码，并配以说明来帮助大家学习本章知识。

13.1 promise 介绍

Introduction to promises

在现实生活中，除了计算机（还有更多东西在那里），我们经常讨论promise（承诺）。让别人做出承诺，或者被别人要求承诺一些事情。直觉上，承诺指的是答应某人的请求再来执行某

个动作。有时候动作会很快进行，有时候必须等待一段时间。在理想世界里，承诺会一直保留。不幸的是，现实世界与理想情况相差甚远，所以有时候承诺无法兑现，不管什么原因。无论什么时候执行和最终的结果，承诺都不会阻止我们同时做其他的事情，比如阅读、做饭或工作。

在JavaScript语言中，promise（承诺）就是这个意思。我们从另一个网站请求资源，结果是服务器的长时间计算值或者JavaScript函数，或者REST服务¹的应答消息（这些都是承诺的例子），在等待结果的时候处理别的任务。当后者成为可用（promise（承诺）是解决/履行）或者请求失败（promise（承诺）是failed/rejected）时，再采取相应的行动。

前面的描述是希望大家完全理解promise（承诺）的概念。但是，还有很多正式的定义。promise（承诺）被广泛讨论，而且这种讨论最后归结为两个提案：Promises/A和Promises/A+。在深入学习jQuery如何处理promise之前，有必要先来了解这两个概念，这样会有更好的理解。

Promises/A+的规范可以在<http://promisesaplus.com/>中找到，但是总的来说，它是这样定义promise的：

promise（承诺）代表异步操作执行的最终结果。与 promise 交互的主要方法就是通过它的 then 方法，该方法可以用来注册回调方法来接受 promise 的最终结果或者为什么 promise 无法兑现。

359

Promises/A+规范

Promises/A+规范定义的then()方法是promise的核心。then()方法接收两个函数：一个在promise完成事件里执行，另外一个在promise被拒绝的时候执行。当promise为其中一种情况时，它都为settled状态。如果promise既不是完成也不是拒绝（比如继续等待服务结果），那就是pending（挂起）状态。

尽管官方文档提到了异步操作，但值得注意的是，promise仍然可以被异步操作解析或者拒绝，所以不仅可以在Ajax请求时使用promise，还有许多其他的情况要在本章里详细介绍。

有时候，操作无论是异步还是同步，可能已经完成了，因此返回值或者拒绝promise的原因都是可以用的。此时，如果使用then()方法，就可以注册函数，这个函数将会立即执行。这也是promise执行与回调函数响应事件的另外一个重大区别。记住，如果为已经触发的事件再添加工理器，回调函数就不会执行。

既然已经学习了promise和then()方法，现在最关键的就是要理解为什么promise如此强大以及为什么可以在工作中帮助我们。要证明这一点，会讨论并解决一个简单的问题场景，与其他工作中遇到的问题类似。首先，使用目前获取的知识来解决它，比如回调函数；然后通过使用promise来逐步改进代码以实现更好的可读性以及可维护性。

¹ REST 服务，可以理解为轻量级服务接口，使用现有的 HTTP 规范交互。网页和移动 APP 使用较多，Java、PHP、RESTWCF、ASP.NET Web API、Node.js 都可以支持开发 REST 接口。

假设要构建一个Web页面，它要调用一个第三方的名为Randomizer的Web服务，这个服务接口API每次都返回随机数。我们希望页面获取两个数，然后求和。一旦完成，就希望在ID为content的页面元素独立显示计算结果。为此，需要执行两个Ajax请求：同步回调函数（使用一个支持函数）和最后展示结果。最复杂的部分是第二个：两个独立的Ajax请求的同步问题。

讨论的页面代码如列表13.1所示。请注意，执行这个页面不会给你任何结果，因为是向不存在的Randomizer Web服务发送请求，但是代码可以帮助你理解promise的性能。

列表 13.1 使用回调函数实现多个异步操作

```
var number1, number2;

function handler() {
    var $content = $('#content');
    if (number1 === null || number2 === null) {
        $content.text('An error has occurred, try again later.');
```

定义 Ajax 回调调用的函数

```
    } else if (number1 !== undefined && number2 !== undefined) {
        $content.text(number1 + number2);
    }
}

$.ajax(http://www.randomizer.com/number', {
    success: function(data, status) {
        number1 = (status === 'success') ? parseInt(data, 10) : null;
        handler();
    },
    error: function() {
        number1 = null;
        handler();
    }
});

$.ajax('http://www.randomizer.com/number', {
    success: function(data, status) {
        number2 = (status === 'success') ? parseInt(data, 10) : null;
        handler();
    },
    error: function() {
        number2 = null;
        handler();
    }
});
```

在页面上显示结果

执行第一个 Ajax 请求

调用同步回调的支持函数

执行第二个 Ajax 请求

调用同步回调的支持函数

前面列表的代码虽然很简单，但是有一个大问题：需要为每个回调函数引入一个变量。此时只需要两个变量，但是随着项目的增大，情况会变得更加复杂，无法管理。此外，假设之前的代码为另外一个函数foo调用的函数的代码体部分，那又怎么返回两个Ajax请求的结果给foo函数呢？使用当前的方法，无法引入全局变量。最后，如果要发送两个Ajax请求，则第二个请求必须在第一个请求完成之后开始。此时，需要在一个回调函数里再添加一个回调函数。越来越多的回调函数的加入，代码会变得非常混乱，这就是我们所熟知的“回调地狱（callback hell）”。

讨论这个例子就是让大家明白回调函数的问题。可以使用promise来改进现有的代码，它有很多好处。ECMA International 小组(<http://www.ecma-international.org/>), JavaScript规范背后的制定者,已经决定在最新版本的JavaScript规范(ECMAScript 2015,也就是ECMAScript 6)里引入promise的概念,并且支持Promises/A+。新的浏览器会提供支持,但是旧的浏览器不会。因此,如果想通过promise方法编写干净的、可读的、可维护的代码,就需要依赖于polyfill或者比标准提供更多函数的库。

虽然jQuery帮助我们避免了所有这些问题,但是取决于使用的版本,它的处理方式可能不同。jQuery提供了Deferred和Promise两个对象,可以在项目里使用它们。但是在介绍它们之前,需要分开讨论,以便于大家掌握本章的这些概念。

jQuery 1.x和jQuery 2.x实现代码遵守了CommonJS Promises/A规范(<http://wiki.commonjs.org/wiki/Promises/A>),它是Promises/A+的基础。因此,在JavaScript和 jQuery 1.x和jQuery 2.x中,使用promise有很多差别。此外,因为jQuery遵守了分支中一个不同的规范,所以这个库无法与其他库兼容。Promises/A定义promise的方式如下:

promise 代表从单个完成操作里返回的最终结果。promise 可能有三种状态,即未完成、完成和失败。

promise状态只能从未完成到完成,或者从未完成到失败。

Promises/A 规范

正如我们看到的,Promise对象的术语定义如图13.1所示,是有点不同。Promises/A定义了未完成、完成和失败三种状态,而Promises/A+定义了挂起、完成和拒绝三种状态。

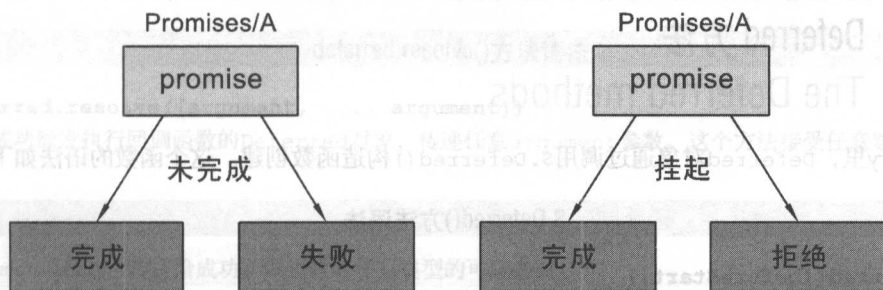


图 13.1 Promises/A 与 Promises/A+中术语定义的差别

这个提案列出了promise的行为,但是没有提供具体的实现代码,所以各种JavaScript库可能都有promise的共同方法then(),但是其他方法可能存在差别。

在jQuery 3.x中兼容标准的promise (ECMAScript 2015, 遵守Promises/A+)。then()方法签名的向后兼容性还有一点不同,但是行为一致。一开始不要担心这些差别,我们会强调所有的不同,而且会提供许多例子代码作为参考。

第13.2节将会学习Deferred和Promise对象的更多内容，需要的时候会强调Promises/A和Promises/A+的其他不同点。

13.2 Deferred 与 Promise 对象

The Deferred and Promise objects

Deferred对象在jQuery 1.5中引入，作为注册多个回调函数到队列中的链式工具，它调用回调队列、依赖于任意同步或者异步请求函数返回的成功或者失败状态。从那时开始，它已经成了讨论的主题，以及匹配的对象，并且一直在修改完善¹。这个对象可以用来执行许多异步操作，比如Ajax请求和动画太小，但是也可以与JavaScript的时间函数一起使用。与Promise对象一起使用，它表示jQuery实现的promise（承诺）概念。

Promise对象从Deferred对象或JavaScript对象创建，拥有Deferred对象的方法子集（always()、done()、fail()、state()、then()）。Deferred对象典型的使用场景就是编写处理异步回调的自定义函数。所以，我们的函数是返回值的生产者（producer），而且希望阻止用户修改Deferred的状态。

第10章介绍了jQuery提供处理Ajax请求的工具函数，但是为了方便，忽略了介绍这些函数的返回值，就是jqXHR对象，大家可能还记得实现的Promise接口。因此，有时候引用Promise兼容的对象，可以调用Promise对象的通用方法。这允许我们编写更加干净和更加可读的代码。

现在来讨论如何使用Deferreds和Promises。

13.3 Deferred 方法

The Deferred methods

在jQuery里，Deferred对象通过调用\$.Deferred()构造函数创建。这个函数的语法如下。

\$.Deferred()方法语法

\$.Deferred([beforeStart])

一个构造函数，它可以注册多个回调函数到队列中的链式工具，它调用回调队列、依赖于任意同步或者异步请求函数返回的成功或者失败状态。选择接受函数参数在构造函数返回前执行。

¹ <http://bugs.jquery.com/ticket/11010>.

“You’re Missing the Point of Promises” by Domenic Denicola: <http://domenic.me/2012/10/14/youremissing-the-point-of-promises/>.

“Javascript promises and why jQuery implementation is broken” by Valerio Gheri: <https://thewayofcode.wordpress.com/tag/jquery-deferred-broken/>.

<https://github.com/jquery/jquery/issues/1722>.

参数

beforeStart(Function) 在构造函数返回之前调用的函数。此函数接受Deferred对象作为函数的上下文(this)。

返回

Deferred对象。

Deferred对象允许链式调用，与其他的jQuery方法相似，但是它还有两个方法。我们永远看不到自己编写这种代码：

```
$.Deferred().html('Promises are great!');
```

363

如果不知道如何使用它，那么创建Deferred对象也没有用处。从第13.3.1节开始，将会介绍这个对象暴露的方法。

13.3.1 解析或者拒绝一个 Deferred 对象

本章第一个讨论的概念就是promise的状态。在jQuery里，promise可以被解析（成功执行）、拒绝（发生错误）或者挂起（既不解析也不拒绝）。这些状态可以通过两种方式获取。第一种是通过代码调用deferred.resolve()、deferred.resolveWith()、deferred.reject()、deferred.rejectWith()方法获取。第二种是通过jQuery函数的失败或者成功获取，例如，\$.ajax()，此时不需要调用之前提到的方法，函数直接返回。

在编写任意使用Deferred对象的代码之前，必须熟悉这些方法，所以先来看看这些方法的语法。

deferred.resolve()方法语法

deferred.resolve([argument, ..., argument])

解析成功触发执行回调函数的Deferred对象，传递任意argument参数。这个方法接受任意数量的参数。

参数

argument(Any) 传递给成功回调函数的任意类型的可选参数。

返回

Deferred对象。

deferred.resolveWith()方法的语法如下。

deferred.resolveWith()方法语法

deferred.resolveWith(context[, argument, ..., argument])

解析成功触发执行回调函数的Deferred对象，传递任意argument参数，设置context作为上下文。

参数

context(Object) 设置作为成功回调函数上下文的对象。

argument(Any) 传递给成功回调函数的任意类型的可选参数。

返回

Deferred对象。

有时候，promise对象需要被拒绝。此时，可以使用deferred.reject()方法。它的语法如下。

deferred.reject()方法语法

deferred.reject([argument, ..., argument])

解析触发执行失败的回调函数的Deferred对象，传递任意argument参数。这个方法接受任意数量的参数。

参数

argument(Any) 传递给失败回调函数的可选参数。

返回

Deferred对象。

与jQuery定义的设置成功回调函数的上下文方法一样，也可以使用deferred.rejectWith()方法来设置失败的回调函数。此方法的语法如下。

deferred.rejectWith()方法语法

deferred.rejectWith(context[, argument, ..., argument])

解析触发执行失败的回调函数的Deferred对象，传递任意argument参数。context作为上下文。

参数

context(Object) 失败回调函数的上下文对象。

argument (Any) 传递给失败回调函数的可选参数。

返回

Deferred对象。

目前为止，已经学习了如何创建Deferred对象，以及如何解析和拒绝它，但是真正的乐趣来自于响应Deferred的状态变化代码。

下面看看如何实现它。

13.3.2 根据解析或者拒绝状态执行函数

通常想知道Deferred对象什么时候被解析来执行一些操作。为此，要使用deferred.done()

方法。它接收一个或者多个参数，可以是一个函数或者函数数组。这些回调函数按照其添加的顺序执行。它的语法如下。

deferred.done()方法语法

deferred.done(callbacks[, callbacks, ..., callbacks])

添加处理器，当Deferred被解析时调用。此方法接受任意数量的回调函数，最少一个。

参数

callbacks(Function|Array) 当Deferred被解析时调用的回调函数或者回调函数数组。

返回

Deferred对象。

与Deferred对象被成功解析时执行的操作一样，也可以在拒绝的时候执行回调函数，使用deferred.fail()方法。它的语法如下。

deferred.fail()方法语法

deferred.fail(callbacks[, callbacks, ..., callbacks])

添加处理器，当Deferred被拒绝时调用。此方法接受任意数量的回调函数，最少一个。

参数

callbacks(Function|Array) 当Deferred被拒绝时调用的回调函数。

返回

Deferred对象。

此时也一样，回调函数的执行与添加时的顺序有关系。

既然已经介绍了这些方法，现在是时候展示它们的实战用法了。开始很简单，修改一下本章最初的简单例子，使用Deferreds对象重构一下代码。这时要开发一个正常工作的demo，所以要引入PHP页面，叫“integer.php”来模拟¹Randomizer服务。例子代码源文件在chapter-13/randomizer.1.html中可以找到。

列表 13.2 使用 Promise 的 Ajax 请求代码，版本 1

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Using Promises with Ajax requests version 1</title>
    <link rel="stylesheet" href="../../../css/main.css" />
  </head>
  <body>
    <p>
```

¹ 模拟随机数服务，很多框架都可以使用，如 ASP.NET MVC、Web API、Java、PHP、Node.js 都行。——译者注


```

The generated random numbers are <span id="number1"></span>
and <span id="number2"></span>.
Their sum is <span id="sum"></span>

<p>

<script src="../../js/jquery-1.11.3.min.js"></script>
<script>
    var number1, number2;

    function handler() {
        var $sum = $('#sum');
        if (number1 === null || number2 === null) {
            $sum.text('An error has occurred, try again later.');
        } else if (number1 !== undefined && number2 !== undefined) {
            $sum.text(number1 + number2);
        }
    }

    var promise1 = $.ajax('integer.php');
    promise1.done(function(data, status, jqXHR) {
        number1 = (status === 'success') ? parseInt(data, 10) : null;
        $('#number1').text(number1);
        handler();
    })
    .fail(function() {
        number1 = null;
        handler();
    });

    var promise2 = $.ajax('integer.php');
    promise2.done(function(data, status, jqXHR) {
        number2 = (status === 'success') ? parseInt(data, 10) : null;
        $('#number2').text(number2);
        handler();
    })
    .fail(function() {
        number2 = null;
        handler();
    });
</script>
</body>
</html>

```

① 存储\$.ajax()返回的 Promise 兼容的对象在一个变量中
 ② 添加一个回调函数，如果第一个 promise 兑现，就执行
 ③ 添加一个回调函数，如果第一个 promise 拒绝，就执行
 ④ 存储\$.ajax()返回的 Promise 兼容的对象在一个变量中
 ⑤ 添加一个回调函数，如果第二个 promise 兑现，就执行
 ⑥ 添加一个回调函数，如果第一个 promise 拒绝，就执行

列表13.2中的代码我们叫版本1，因为需要不断重构代码，直到开发出一种干净的和优雅的解决方案为止。正如大家看到的，这个代码与列表13.1没有太大不同，但是它展示了一些重要的概念。

第一个概念，存储\$.ajax()函数返回的jqXHR对象在promise1变量①和promise2④中。正如我们提到的，jqXHR对象是兼容Promise的，这意味着可以像调用done()和fail()一样调用它。使用变量不是必需的，因为可以直接使用链式调用功能，但是想通过变量名搞清楚\$.ajax()方法返回的对象。

然后可以添加列表13.1里定义的回调函数。如果Ajax请求成功执行，则回调函数会通过调用

done()方法添加到promise1②和promise2⑤变量上。与失败的回调函数添加过程一样,通过调用fail()方法为promise1③和promise2⑥添加。

在讨论done()和fail()方法时,想强调的是,如果要想在Deferred对象修改为解析或者拒绝之后添加成功或者失败的回调函数,则回调函数会立即执行。思考下面的代码:

```
var promise1 = $.ajax('integer.php');
setTimeout(function() {
    promise1.done(function(data, status, jqXHR) {
        //这里编码
    })
    .fail(function() {
        //这里编码
    });
}, 5000);
```

此时通过延迟5秒来模拟“integer.php”页面执行和返回过程(例子模拟实验够了),再添加回调函数。基于这个假设,调用done()和fail()来添加回调函数,promise1已经定义了。我们所期望的,基于对事件侦听器的经验,是它们都不会执行。原因是promise的事件变化已经发生了。但是两个函数都会执行,这也是一个重大的区别。

另外一个有趣的区别就是,可以在单个语句里添加许多个回调函数。假设Ajax请求是成功的,想要执行两个函数foo()和bar(),在传统的方法里,可以像下面一样编写代码:

```
$.ajax('integer.php', {
    success: function(data, status, jqXHR) {
        foo(data, status, jqXHR);
        bar(data, status, jqXHR);
    }
});
```

使用done()方法可以在单行代码里添加两个函数:

```
$.ajax('integer.php').done(foo, bar);
```

或者等价于(注意这里传递的是函数数组):

```
$.ajax('integer.php').done([foo, bar]);
```

更加完美,是不是?

列表13.2使用了一些新的方法,但是仍然存在使用同步方法的问题。下面来看看如何优化它。

◀ 368

13.3.3 when()方法

为了编辑列表13.2为最终版本,需要介绍另外一个工具函数\$.when()。它提供了一种根据一个或者多个对象执行回调函数的简单方法,通常对象与Deferred或者Promise兼容。这就是

我们需要的代码，因为通过两个`$.ajax()`调用返回了两个Promise兼容的对象。但是在使用它之前，先来看看它的语法和参数。

`$.when()`方法语法

`$.when(object[, object, ..., object])`

提供了一种方法，可以根据一个或者多个对象来执行回调函数，通常用于表示异步事件的Deferred或者Promise兼容的对象。这个函数接受任意数量的对象参数，至少有一个。

参数

`object(Deferred|Promise|Object)` 一个Deferred、Promise、Promise兼容的或者JavaScript对象。如果传递JavaScript对象，则它会作为Deferred对象处理。

返回

Promise 对象。

`$.when()`工具函数有一个有趣的地方值得强调：它不会返回Deferred对象而是返回Promise对象。此工具函数的返回对象不能被解析或者拒绝；只能调用`done()`、`fail()`以及一些其他方法。值得注意的是，ECMAScript 2015有一个相似的方法叫`Promise.all()`。

如果传递单个Deferred对象给`$.when()`，则它的Promise对象被返回；否则，会从源Deferred对象中创建一个新的Promise对象来跟踪传递给`$.when()`的所有对象(Promise、Promise兼容的、Deferred对象等)的状态。

当传递给它的所有对象(Promise、Promise兼容的、Deferred对象等)成功解析后，`$.when()`触发执行成功的回调函数（当函数返回成功）。反之，只要传递的对象被拒绝，它就会触发执行失败函数。传递给回调函数的参数，无论成功或者失败，都是传递给`resolve()`或者`reject()`的参数，这取决于实际情况。

在感到迷惑之前，先来看一个具体的例子。正如之前提到的，我们会重写列表13.2，而且会使用Deferred重构代码。最终的代码文件在chapter-13/randomizer.2.html中。

369

列表 13.3 使用 Promise 的 Ajax 请求，版本 2

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Using Promises with Ajax requests version 2</title>
    <link rel="stylesheet" href="../css/main.css" />
  </head>
  <body>
    <p>
      The generated random numbers are <span id="number1"></span>
      and <span id="number2"></span>.
      Their sum is <span id="sum"></span>
    </p>
```

```

<script src="../../js/jquery-1.11.3.min.js"></script>
<script>
    function success(params1, params2) {
        var number1 = (params1[1] === 'success') ?
            parseInt(params1[0], 10) : null;

        var number2 = (params2[1] === 'success') ?
            parseInt(params2[0], 10) : null;

        if (number1 === null || number2 === null) {
            fail();
            return;
        }

        $('#number1').text(number1);
        $('#number2').text(number2);
        $('#sum').text(number1 + number2);
    }

    function fail() {
        $('#sum').text('An error has occurred, try again later.');
```

通过\$.when() ①
从内部创建的
“master”
Deferred 来创
建新的
Promise, 根据
\$.ajax()调用

② 定义成功回调函数

③ 定义失败回调函数

④ 使用 deferred.done()调
用 success()回调函数

⑤ 使用 deferred.fail()
设置 fail()回调函数

```

    $.when($.ajax('integer.php'), $.ajax('integer.php'))
        .done(success)
        .fail(fail);
    </script>
</body>
</html>

```

直接查看代码应该会感觉轻松愉快。由于使用了\$.when(), 所以大大简化了列表13.2的代码, 代码变得更加可读和可管理。下面来分析一下代码。

使用\$.when()的关键点, 就是用来解决Ajax请求结果同步的问题③。通过它, 不必为每个成功或者失败的函数设置函数。事实上, 只需要为\$.when()返回的Promise对象设置即可, 使用done()④和fail()⑤。再说一遍, Promise对象是从Deferred对象或者其他的jQuery对象(假设其通过\$.when()创建)创建的, 而且拥有了Deferred的方法子集(always()、done()、fail()、state()、then())。

当两个Promise兼容的对象完成时, success()函数才会执行。它的行为与之前的版本相比没有变化, 但是有一个有趣的地方值得讨论。定义了params1和params2两个参数给这个函数, 因为这是我们使用的Promise兼容的对象个数①。每个参数都是一个包含要传递给\$.ajax()、\$.get()、\$.post()的成功回调函数的参数, 即data、statusText、jqXHR。值得注意的是, 如果传递的值是一个对象, 则它不会被包装。

列表中定义的最后一个函数为fail()⑤。它是从之前的列表的handler()函数中抽取出来的, 它在Ajax执行失败的时候会立即执行。

除了解析和拒绝Deferred对象, 我们也可以通知执行过程的进度。

13.3.4 通知 Deferred 的进度

有时候,可能需要知道Deferred的状态。例如,如果要异步查询数据,那么要知道完成的百分比进度。如果过程是基于promise的,则可能有一个函数等待这个promise的解析或者拒绝结果,你要保持通知的状态。此时,需要使用deferred.notify()。它的语法如下。

deferred.notify()方法语法

deferred.notify([argument, ..., argument])

传递一个参数argument,触发任意进度的回调函数。这个方法接受任意数量的参数。

参数

argument(Any) 任意类型的可选参数,它被传递给定义的回调函数。

返回

Deferred对象。

371 加入想强迫设置执行回调函数的上下文,可以使用deferred.notifyWith()方法。

deferred.notifyWith()方法语法

deferred.notifyWith(context[, argument, ..., argument])

触发执行任意进度的回调函数,传递argument参数,设置它们的上下文context。

参数

context(Object) 进度回调函数的上下文。

argument(Any) 任意类型的可选参数,它被传递给过程回调函数。

返回

Deferred对象。

有了这些方法,现在可以看看在过程中执行什么操作了。

13.3.5 跟踪进度

有了之前讨论的方法,可以通知异步操作的进度了。但是,如果不能侦听这些更新,那么毫无意义。现在需要deferred.progress()方法帮忙。

deferred.progress()方法语法

deferred.progress(callbacks[, callbacks, ..., callbacks])

当Deferred对象接到进度提醒时,添加调用的处理器函数。这个方法接受至少一个回调函数作为参数。

参数

callbacks(Function|Array) 当Deferred对象接到进度通知的时候,调用一个函数或者函数数组。

[返回](#) [在 C# 中调用 COM 方法来解析](#)

Deferred对象。

现在既然已经知道这个方法如何工作，现在来看看具体的实战例子。假设需要创建一个动画特效的进度条，再使用进度条来更新完成的百分比。为此，需要使用`deferred.progress()`和`deferred.notify()`方法。

注意：要开发的例子不是理想化的。一个更好的办法就是返回使用的 Deferred 对象的 Promise 对象，让动画函数的调用者更新百分比。我们会在第 13.3.6 节里修改代码，但是此时想尽量简单。

前面需求的代码可以在JS Bin (<http://jsbin.com/yohiho/edit?html,css,js,output>)中找到。也可以在本书源码chapter-13/deferred.progress.1.html里找到，如列表13.4所示。

列表 13.4 使用 deferred.progress(), 版本 1

```

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Using deferred.progress() version 1</title>
<link rel="stylesheet" href="../../../css/main.css" />
<style>
    .bar,
    .inner-bar
    {
        height: 50px;
        border-radius: 3px;
    }
    .bar
    {
        position: relative;
        border: 1px solid #000000;
        background-color: #CCCCCC;
    }
    .inner-bar
    {
        width: 0%;
        background-color: #F72F39;
    }
    .progress
    {
        position: absolute;
        font-size: 30px;
        width: 100%;
        text-align: center;
        top: 10px;
    }
    button

```



```

15.3.4 {
    margin-top: 15px;
}
</style>
</head>
<body>
    <div class="bar">
        <div class="inner-bar"></div>
        <span class="progress">0%</span>
    </div>
    <button id="run-button">Run animation</button>

    <script src="../../js/jquery-1.11.3.min.js"></script>
    <script>
        function animate(milliseconds) {
            var $innerBar = $('<div class="inner-bar"></div>').width(0);
            var $barWidth = $('<div class="bar">').width();
            var step = $barWidth / 100;
            var deferred = $.Deferred().progress(function (value) {
                $('<span class="progress">').text(Math.floor(value) + '%');
            });

            var idInterval = setInterval(
                function () {
                    // 不能用 width() 检索宽度
                    // 因为 #1724 问题
                    // (https://github.com/jquery/jquery/issues/1724)
                    var nextWidth =
                        parseFloat($innerBar.get(0).style.width) + step;
                    deferred.notify(nextWidth * 100 / $barWidth);
                    $innerBar.width(nextWidth);
                    if (nextWidth >= $barWidth) {
                        deferred.resolve();
                        clearInterval(idInterval);
                    }
                },
                milliseconds / 100
            );

            $('#run-button').click(function() {
                animate(1000);
            });
        }
    </script>
</body>
</html>

```

① 定义一个简单的进度条

② 定义进度条动画函数

动画的每一步都通知 Deferred，传递动画完成度

③ 当动画完成时解析 Deferred

373 创建新的 Deferred，给进度条添加回调函数

这个例子里我们创建了一个简单的进度条来展示完成的百分比进度①。在script元素内部，定义了展示动画进度条的函数animate()②。该函数接收一个毫秒数字来控制动画的显示时间。在函数内部，实例化了一个新的Deferred对象，添加了一个进度回调函数来更新完成的进度③。

使用JavaScript的setInterval()函数，设置animate()函数的核心部分，计算下一步动画，每次都把进度条宽度增加\$barWidth/100，并且通知Deferred④。最后当动画完成时，使用

`deferred.resolve()` 方法来解析Deferred对象⑤。

这个例子展示了如何使用`deferred.progress()`方法。但是可能你还好奇为什么不在传递给`setInterval()`的回调函数内部更新百分比，并且去掉Deferred对象。实际上可以这么做，但是前面的列表已经给了我们机会走向一个更重要的概念：什么时候使用Deferred对象或者Promise对象，以及为什么。

13.3.6 使用 Promise 对象

为了精通Deferreds 和Promises，需要知道什么时候使用哪个对象。为了帮助大家理解这个主题，假设要实现一个基于promise的超时函数。我们就是函数的制造商（producer）。此时，函数的消费者（consumer）不需要关心解析或拒绝它。在完成、失败或者Deferred的过程中，消费者只需要添加处理器来执行操作。此外，还需要确保消费者不能随意解析或者拒绝Deferred对象。 <374

为此，需要在超时函数里返回Deferred的Promise对象，而不是Promise本身。为了实现这个目标，下面介绍一个新的方法`deferred.promise()`。

deferred.promise()方法语法

deferred.promise([target])

返回Deferred的Promise对象。

参数

`target(Object)` promise方法要附加的对象。如果提供这个参数，此方法附加Promise的行为给此对象，并返回对象，而不是创建新的对象。

返回

Promise对象。

既然已经知道了这个方法，现在使用它来编写一些实战的例子代码。

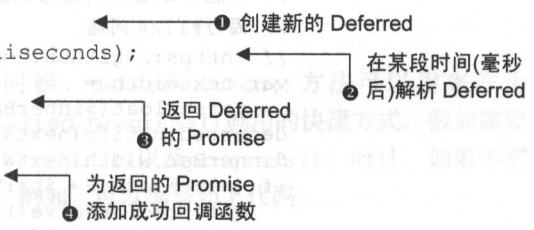
代码如列表13.5所示，在JS Bin(<http://jsbin.com/kefaza/edit?js,output>)和chapter-13/promise.timer.html文件里都可以找到。

列表 13.5 基于 promise 的定时器

```
function timeout(milliseconds) {
  var deferred = $.Deferred();
  setTimeout(deferred.resolve, milliseconds);

  return deferred.promise();
}

timeout(1000).done(function() {
  alert('I waited for 1 second!');
});
```



① 创建新的 Deferred

② 在某段时间(毫秒后)解析 Deferred

③ 返回 Deferred 的 Promise

④ 为返回的 Promise 添加成功回调函数

在列表13.5中定义了`timeout()`函数,它包装了JavaScript的`setTimeout()`函数。在`timeout()`函数内部,创建了一个新`Deferred`对象(我们是生产者)❶,组织`setTimeout()`在一定时间后解析`Deferred`对象❷。一旦完成,就返回`Deferred`的`Promise`对象❸。这样做,可以确保函数的调用者(消费者)❹无法解析或者拒绝`Deferred`对象,而只能够使用`deferred.done()`和`deferred.fail()`来添加回调函数。

之前的例子十分简单,可能还没有完全澄清何时使用`Deferred`何时使用`Promise`。对于还有疑问的读者,下面来讨论另外一个例子。我们会重构进度条demo, `animate()`函数只专注于更新进度条,从更新文本展示进度百分比的工作中解放出来。因此,函数的调用者必须更新百分比,或者当`Deferred`对象被解析或者拒绝的时候执行其他任务。

新版本的代码如列表13.6所示,也可以在JS Bin(<http://jsbin.com/cefece/edit?html,css,js,output>)和chapter-13/deferred.progress.2.html中找到。注意,列表中忽略了页面样式代码,这样可以只专注于代码。此外,粗体显示了修改的代码。

列表 13.6 使用 `deferred.progress()`, 版本 2

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Using deferred.progress() version 2</title>
    <link rel="stylesheet" href="../css/main.css" />
  </head>
  <body>
    <div class="bar">
      <div class="inner-bar"></div>
      <span class="progress">0%</span>
    </div>

    <button id="run-button">Run animation</button>

    <script src="../js/jquery-1.11.3.min.js"></script>
    <script>
      function animate(milliseconds) {
        var $innerBar = $('inner-bar').width(0);
        var $barWidth = $('bar').width();
        var step = $barWidth / 100;

        var deferred = $.Deferred();
        var idInterval = setInterval(
          function () {
            // 不能用 width() 检索宽度
            // 因为#1724 问题
            // (https://github.com/jquery/jquery/issues/1724)
            var nextWidth =
              parseFloat($innerBar.get(0).style.width) + step;
            deferred.notify(nextWidth * 100 / $barWidth);
            $innerBar.width(nextWidth);
            if (nextWidth >= $barWidth) {
              deferred.resolve();
              clearInterval(idInterval);
            }
          }
        );
      }
    </script>
  </body>
</html>
```

❶ 创建新的 `Deferred`

```

    },
    milliseconds / 100
  );
  return deferred.promise();
}

$('#run-button').click(function() {
  animate(1000)
    .progress(function(value) {
      $('#progress').text(Math.floor(value) + '%');
    })
    .done(function() {
      alert('The process is completed');
    });
});
</script>
</body>
</html>

```

② 返回 Deferred 的 Promise

添加当返回 Promise 过程中执行的处理器

添加当返回 Promise 解析中执行的处理器

在列表13.6里，我们看到`animate()`只创建了Deferred①。一旦代码运行动画，就会返回Promise，这样调用者就可以为它的方法添加回调函数了②。再说一次，返回Promise对象，是为了让调用者可以添加回调函数，但是不要修改Deferred的状态。此时，它的职责就是创建Deferred对象。

最后，在按钮点击事件的处理器内部，使用`deferred.progress()`③添加一个回调函数，在动画进度过程中运行，然后当Deferred解析的时候使用`deferred.done()`（上一个版本里没有）④执行回调函数。

有了这个例子，希望给大家强化了目前介绍的概念和方法。最细心的读者可能注意到，Promises/A 和Promises/A+规范都介绍了，还有一个重要的方法没有介绍。下面我们来看看。

13.3.7 使用 then()简化代码

第13.1节总结了Promises/A和Promises/A+规范定义的promise。这两个规范都提到了`then()`方法，但是Promises/A规定它必须接受第三个参数，这个参数用来作为进度事件的参数；而Promises/A+没有这个参数。jQuery在版本3(版本1.x和2.x)之前实现了这个方法，它与两个规范都不同，因为它定义的第一个参数——成功回调函数，是强制的；其他两个参数，失败和进度回调函数，是可选的。与之相反，Promises/A 和Promises/A+规范规定所有的参数都是可选的。

当只需要为这些方法执行一个处理器的时候，`deferred.then()`方法可以用来作为`deferred.done()`、`deferred.fail()`、`deferred.progress()`调用的快捷方式。假如需要多个处理器，则可以链式调用`then()`、`done()`、`fail()`以及`progress()`。而且，如果不需要特定类型的处理器，则可以简单传递`null`。例如，可以编写如下代码：

377

```
$.Deferred()  
  .then(success1)  
  .then(success2, null, progress1)  
  .done(success3);
```

既然已经知道了这个方法的用途，现在是时候学习它的语法了。

deferred.then()方法语法

```
deferred.then(resolvedCallback[, rejectedCallback  
[, progressCallback]])
```

定义当Deferred对象解析、拒绝或者在过程中执行的处理器。加入参数不是必须的，可以传递null。

参数

resolvedCallback(Function) 当Deferred解析时执行的函数。

rejectedCallback(Function) 当Deferred拒绝时执行的函数。

progressCallback(Function) 当Deferred进行时执行的函数。

返回

Promise 对象。

deferred.then()方法返回Promise对象而不是Deferred。在调用它之后，就无法再解析或者拒绝Deferred对象，除非对其保存一个对象引用。回忆列表13.6里定义的按钮点击事件处理器的代码，其使用了deferred.then()方法，可以重构如下，效果一样：

```
animate(1000).then(  
  function() {  
    alert('The process is completed');  
  },  
  null,  
  function (value) {  
    $('#.progress').text(Math.floor(value) + '%');  
  }  
);
```

让这个方法更加有趣的是，它有能力把接受的值作为参数转发给deferred.then()、deferred.done()、deferred.fail()、deferred.progress()的调用代码，调用代码在这之后定义。

在绝望地开始哭泣之前，来看一个简单的例子：

```
var deferred = $.Deferred();  
deferred  
  .then(function(value) { return value + 1; })  
  .then(function(value) { return value + 2; })  
  .done(function(value) { alert(value); });  
deferred.resolve(0);
```

这些代码创建了一个新的Deferred，然后添加了三个函数，当它被解析的时候再执行，两个

函数使用了`deferred.then()`方法，而一个函数使用了`deferred.done()`方法。最后一行使用`0`来解析Deferred对象，引发执行定义的三个函数（按照添加的顺序）。

在使用`deferred.then()`添加的函数里，返回了一个新的值，是从接收的参数创建的。第一个函数接收到`0`，因为这是传递给`deferred.resolve()`的参数，求和为`1`，然后返回结果。返回的求和结果使用`deferred.then()`传递给下一个函数。第二个函数接收`1`代替`0`作为参数。这个值求和为`2`，返回结果`(3)`被下一个处理器使用。此时，处理器使用`deferred.done()`添加，它没有这个功能，所以最后的值是`3`。

如果使用`deferred.done()`在之前的代码中添加其他函数，从调用链里第三个参数（使用`deferred.done()`添加的）返回一个修改的值，新的处理器将会接受相同的值作为第三个参数。下面的代码将会两次弹出值`3`：

```
var deferred = $.Deferred();
deferred
    .then(function(value) { return value + 1; })
    .then(function(value) { return value + 2; })
    .done(function(value) { alert(value); return value + 5; })
    .done(function(value) { alert(value); });
deferred.resolve(0);
```

Promises/A 和 Promises/A+兼容的库中（例如jQuery 3.x），抛出异常会转换为拒绝，失败函数会使用异常执行。在jQuery 1.x和2.x中，未捕获的异常会阻止程序的执行。思考下面的代码：

```
var deferred = $.Deferred()
deferred
    .then(function(value) {
        throw new Error('An error message');
    })
    .fail(function(value) {
        alert('Error');
    });
deferred.resolve();
```

在jQuery 3.x里，会看到弹出消息“Error”。相反，jQuery 1.x 和2.x将会冒泡抛出异常，通常会到达`window.onerror`。如果没有定义函数，则会看到控制台显示消息“Uncaught Error: An error message”（未捕获的错误：一个错误消息）。

也可以更进一步学习这个问题，以了解不同的行为。看下面的代码：

```
var deferred = $.Deferred();
deferred
    .then(function() {
        throw new Error('An error message');
    })
    .then(
        function() {
            console.log('First success function');
        },
        function() {
            console.log('First failure function');
```



```

    }
    .then(
        function() {
            console.log('Second success function');
        },
        function() {
            console.log('Second failure function');
        }
    );
deferred.resolve();

```

在jQuery 3.x里，这些代码会在控制台输出消息“First failure function”和“Second success function”。原因是，我们提到过规范抛出的异常会转换为拒绝，使用异常来调用失败回调函数。此外，一旦异常被失败回调函数处理（then()添加的函数），下面成功的函数就应该会执行（这个例子里，成功回调函数传递给第三个then()）。

在jQuery 1.x和2.x里，除了第一个函数（抛出错误）执行外，其他函数都不执行。只会在控制台看到“Uncaught Error: An error message”（未捕获的错误：一个错误消息）。

jQuery 3: 添加的方法

jQuery 3添加了新的方法给Deferred和Promise对象，叫catch()。它定义了当Deferred和Promise被拒绝时执行的函数处理器。它的签名如下：

```
deferred.catch(rejectedCallback)
```

此方法只是then(null、rejectedCallback)的简化，而且已经添加到jQuery 3中，ECMAScript 2015规范里有一个同名的方法。

380 > 尽管jQuery与规范有一些不同，Deferred仍然是一个强大的工具。作为高级开发者，而且随着项目难度的增加，我们发现经常要使用它。

有时候，无论Deferred的状态是什么，都想要执行一个或者多个操作。jQuery提供了针对这种情况的方法。

13.3.8 一直执行处理器

也可能当想要一直执行一个或者多个处理器的时候，不考虑Deferred的状态。为此，可以使用deferred.always()方法。

deferred.always()方法语法

```
deferred.always(callbacks[, callbacks, ..., callbacks])
```

添加调用的处理器，当Deferred对象或解析或拒绝时调用。它接收任意数量的回调函数，最少一个。

参数

callbacks(Function|Array) 当Deferred对象解析或拒绝时调用的回调函数，或者函数数组。

返回

Deferred对象。

思考下面的代码：

```
var deferred = $.Deferred();
deferred
  .then(
    function(value) {
      console.log('success: ' + value);
    },
    function(value) {
      console.log('fail: ' + value);
    }
  )
  .always(function() {
    console.log('I am always logged');
  });
deferred.reject('An error');
```

当执行以上代码时，会在控制台看到两个消息。第一个是“fail: Anerror”，因为Deferred已经被拒绝。第二个是“I am always logged”，因为使用deferred.always()添加的回调函数被执行了，无论Deferred的状态是什么。

还有最后一个要讨论的方法。

13.3.9 确定 Deferred 的状态

当使用Deferred编写代码时，也许需要去检验它的当前状态。为此，可以使用deferred.state()方法。它就是我们要找的工具：返回一个字符串来指定当前的Deferred状态。它的语法如下。

381

deferred.state()方法语法

deferred.state()

确定Deferred对象的当前状态。返回一个字符串，值为“resolved”、“rejected”、“pending”之一。

返回

一个表示Deferred状态的字符串。

这个方法在进行单元测试时非常有用。例如，可以编写如下语句。

```
assert.equal(deferred.state(), 'resolved');
```

前面语句里使用的这个方法来自于QUnit单元测试框架，我们将会在第14章里讨论。它用于简单验证第一个参数是否与第二个参数相等，此时测试通过。

13.4 promise 化一切

Promisifying all the things

前面的章节专注于Deferred对象和Promise对象，以及它们的方法，但是也保留了另外一个小惊喜：`promise()`方法。该方法与我们之前学习的`deferred.promise()`方法不同。`promise()`方法允许我们转换jQuery对象为Promise对象，可以使用本章讨论的方法来添加处理器。它的语法如下。

promise()方法语法

promise([type][, target])

返回一个动态生成的Promise对象，一旦某个类型全部绑定到集合、队列或者没有队列的操作执行完毕，它就会被解析。默认类型是fx，这意味着当所有选择元素的动画执行完毕后，返回的Promise对象才被解析。

参数

type(String) 观察队列的类型。默认值为fx，表示特效的默认队列。

target(Object) `promise()`方法要附加到的对象。

返回

Promise对象。

根据方法的描述，如果使用的jQuery对象没有运行动画，它就会被立即解析。因此，任意附加的处理器被立即执行。思考下面的代码：

```
$('#p')
    .promise()
    .then(function(value) { console.log(value); });
```

如果页面的代码不运行任意动画，使用`then()`添加的函数就会立即执行，传递给它的值就是jQuery对象本身。

现在思考一个更复杂的例子，它使用了第8章里介绍的`animate()`方法来创建动画。在这个例子里，将会创建两个正方形，然后从左向右以不同的速度移动它，所以，动画会在不同的时间结束。一旦动画效果结束，就会弹出一个成功消息。

实现此目标的代码如列表13.7所示，也可以在下面地址找到源代码：JS Bin(<http://jsbin.com/wuhiqa/edit?js,output>)及chapter-13/promisify.html。

列表 13.7 Promise 化一个 jQuery 集合

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Promisifying a jQuery object</title>
    <link rel="stylesheet" href="../css/main.css" />
    <style>
      .square
      {
        position: relative;
        width: 25px;
        height: 25px;
        background-color: #1E39BC;
        margin: 10px 0;
      }
    </style>
  </head>
  <body>
    <div id="square1" class="square"></div>
    <div id="square2" class="square"></div>

    <script src="../js/jquery-1.11.3.min.js"></script>
    <script>
      动画第一 ①
      个方形  →  $('#square1').animate({left: 500}, 1500);
      动画第二 ②
      个方形  →  $('#square2').animate({left: 500}, 3000);
                  ③ 基于包含两个正
                  方形的 jQuery 对
                  象创建 Promise
                  .promise()
                  .done(function() {
                    alert('The animations are completed');
                  });
                  ④ 定义一个函数，一
                  旦两个动画完成，
                  就执行
    </script>
  </body>
</html>
```

在以上代码里，动画了两个正方形，但是第一个动画会持续1500毫秒①，而第二个动画会持续3000毫秒。在这些动画定义完成以后，要使用它们共享的样式名来选择正方形元素，使用本节③介绍的promise()方法来创建Promise对象。最后，添加一个函数，在两个动画都完成的时候执行它④。酷不酷？帅不帅？

◀ 383

13.5 总结

Summary

本章介绍了promise（承诺）的概念，一种处理异步操作的更好的模式。promise可以避免使用讨厌的方法来同步并行的异步函数，以及在回调函数里不停地嵌套回调函数。

还介绍了Promises/A 和 Promises/A+规范，对比了与jQuery版本promise的不同点。也解释了两个协议的不同之处，关注在采用的术语上，以及then()的参数数量、promise的核心知识。

jQuery通过两个对象Deferred和Promise实现了promise。正如我们描述的,后者从Deferred或者jQuery对象创建,拥有Deferred方法的子集。

也详细介绍了Deferred暴露的方法。可以使用deferred.resolve()解析,或者使用deferred.reject()拒绝。也可以使用deferred.notify()方法通知异步操作的进度。所有这些方法都有一个关联方法,它接收一个额外的参数,允许设置触发的上下文对象: deferred.resolveWith(), deferred.rejectWith(), deferred.notifyWith()。

deferred.done()、deferred.fail()和deferred.progress()是允许我们添加在Deferred对象被解析、拒绝或者进行中时执行的方法。

另外一个有趣的概念就是\$.when(),这个工具函数允许我们简便地同步多个异步或者同步函数。

我们介绍了jQuery实现的then()方法,强调了jQuery库与其他实现Promises/A或者Promises/A+库的不同点。

本章也介绍了Deferred对象暴露的其他方法,例如deferred.always()和deferred.state()。

最后,讨论了promise()方法,它允许我们把任意的jQuery对象转换为Promise对象。此方法非常有用,因为它让jQuery对象具备了Deferred对象的强大功能。

结束本章内容时我们完全掌握了jQuery库。有了目前为止大家学习的内容,希望各位了解了这个库的底层结构、基本原理及扩展接口。现在可以定义自己为一个jQuery高手了!恭喜!

第14章将会尝试更上一层楼,讨论另外一个重要的概念,每个高级开发者都会遇到的问题:

384 unit testing (单元测试)。下面继续学习。

使用QUnit进行单元测试

Unit testing with QUnit

本章内容

- 为什么测试很重要？
- 什么是单元测试？
- 安装QUnit。
- 使用QUnit测试JavaScript代码。
- 如何使用QUnit创建完成的测试套件。

第13章讨论的概念与jQuery核心紧密关联。Deferred和Promise可能已经足够困难，但是希望有了本书的帮助，大家学习起来顺利一些。

现在是时候再强化我们的技术了。本章将会学习其他必须知道的更高级的工具和技巧。我们会在使用jQuery编写代码的时候使用它们，但是可以在JavaScript编写的代码里重用这些知识。如果大家在团队，而不是仅仅为个人使用，测试是要求必须掌握的核心概念。

本章学习的重要主题是测试，以及如何做出断言。断言是检验代码运行正常，并且与期望值相等的一种表达方式。另外一种表述方式，断言检验函数的返回值是否与期望值相等，或者变量的值或者对象的属性是否是我们期望的值。

最后，在简要介绍完单元测试和为什么要思考它之后，还要再讨论QUnit。这是JavaScript社区里提供单元测试JavaScript代码的工具。讨论QUnit的原因是，QUnit是jQuery项目(jQuery、jQuery UI、jQuery Mobile)官方单元测试的工具框架。

长话短说，现在就来看看测试的知识，特别是单元测试和QUnit。

14.1 为什么测试很重要

Why is testing important?

要解释为什么软件测试非常重要，就从一个例子开发吧。如果有人要你开一辆没有在F1方程式赛车里测试过的汽车，你会怎么回答？你会为一个无人验证是否安全的冒险玩命尝试吗？如果汽车在第一个弯道上损坏了怎么办？或者说，实际上刹车刹不住？当然，这些问题的回

< 385

答都是大大的“不”字，同样的原则适用于软件系统。没有人愿意使用每两分钟就崩溃一次的软件，或者无法按预期工作的软件，甚至更糟糕，无法满足用户的需求。这也是为什么引入软件测试的原因。

软件测试是一个使用一些输入参数评估软件细节来发现实际输出结果和期望之间差别的过程。测试可帮助构建安全、可靠和稳定的软件系统。它提供了有价值的信息和软件系统状态的内部细节。软件测试也可以作为衡量软件产品与需求规范、用户期望或者旧版本产品差别的标准。另外，一个主要的目的是查找代码的错误，通常称为bug（缺陷）。

当讨论bug探测时，我们不是想检查所有可能的条件，而是查找系统中所有的bug，这在实际系统中是不可能完成的。这不是使用方法或者知识可以解决的问题，而是每个复杂系统都存在这个问题。我们必须接受系统中存在的bug，不是因为我们和其他开发人员还不够优秀，而是因为任何实际的项目都非常复杂，无法预测，并修复所有的软件bug。

测试是开发人员生命的一部分。至少，应该是。不幸的是，许多开发者害怕测试。测试被当成多余的工作，一些浪费时间的事情——许多时间。当然，我们不会走向极端。如果正在开发真实的小项目，一生中就执行一次，那么测试可能不值得。但是作为有经验的开发者，如果正在开发的项目或者库将来都会使用，甚至与团队共享，或者为整个JavaScript社区共用，

386 > 那最好还是测试一下。

测试是一门非常广泛的学科。我们可以测试项目的许多方面（例如，兼容性测试和回归测试），使用不同的方法（例如，可视化测试和黑盒测试）和不同的级别（例如，单元测试和集成测试）。但是，本节的目标不是教给大家软件测试的目的。主题太广泛，可能需要一本书来介绍。我们要说的是测试的重要性以及为什么要测试自己的软件代码。

第14.1.1节将会给大家介绍提到的测试类型之一：单元测试。

14.1.1 为什么要单元测试

单元测试就是把软件分割成不同的片段，成为单元（units），进行独立测试，检验单元代码是否符合预期的软件测试方法。当使用单元测试时，每个单元测试的目标应该相互独立。通常，一个单元表示为一个函数或方法，具体依赖于编程语言的特性。

简而言之，单元测试的主要好处如下。

- 验证代码根据输入数据返回期望的结果。
- 早期发现重大缺陷（与前一点相关联）。
- 改善代码的设计。
- 标识复杂的单元。

遵守单元测试的原则，给函数一个输入参数的集合，就可以确定函数是否返回期望的值。这

这个过程通常都是自动化的，而且涉及单元测试框架。单元测试由编写的函数组成，传递各种输入的参数数据给目标测试函数。这些单元测试函数会检验测试函数返回输出结果与期望值。使用这个方法，也可以检查无效的输入参数，目标函数能够优雅地处理（这意味着产生错误或者抛出异常）。当一个或多个测试失败时，可以知道单元里有错误，需要修复它。这个过程会持续迭代直到所有的单元测试全部通过。

单元测试的目标是在软件开发的早期阶段尽可能多地发现软件开发的bug。另外一个好处就是，一旦所有的单元测试代码编写完成，就有理由来改进现有的单元代码（函数或者方法）。如果更新代码出错，一个或者多个测试会失败，就知道又出问题了。因此，可以更自信地修改而不会影响已经正常的功能。

另外一个单元测试的好处就是，可以帮助我们理解和改善代码的设计。必须从一开始列出要满足的条件和输出结果，而不是编写代码做什么。这个概念通常与开发术语相关，称为测试驱动的开发（test-driven development, TDD）。

测试驱动的开发

测试驱动的开发是一个依赖于在开发之前先编写测试单元的代码。测试驱动的开发中的第一步是编写失败的测试用例。然后开发者必须开发出实现功能的代码，直到所有的功能完成。最后，代码持续重构直到满足可以接受的质量需求标准。

最后一个单元测试的好处是，可以帮助我们辨识复杂的代码单元。例如，可以区分一个方法做了比最初的目标更多的工作——记住单一职责原则(SRP)¹。通常，当编写测试代码的时候会感到过于复杂或者太长。这种情况下就应该考虑分析或重构代码了。

如果项目使用JavaScript开发，还有一个原因：浏览器的兼容性。虽然主流的浏览器，包括IE，都遵守越来越新的规范，但是它们在不同的环境中仍然还有许多差别。强壮的测试是避免某些特定浏览器工作的代码在部署后出现问题的方式之一。可以在不同浏览器里运行相同的测试来检验是否所有的用例都能正常通过。

现在，既然已经知道什么是测试和单元测试，而且了解测试的重要作用，现在是时候看一下可用的JavaScript单元测试框架了。

14.1.2 JavaScript 单元测试框架

知道那个笑话吗？在JavaScript开发者中非常流行的，说的是，你想到一个词，然后谷歌搜索

¹ OOAD（面向对象分析与设计）的五大原则之一。单一职责原则(SRP)指的是：一个对象应该只包含单一的职责，并且该职责被完整地封装在一个类中。这也是高内聚的体现。我们在高级架构师训练营课程中也有强调。——译者注

“<词>.js”，如果该词对应的JavaScript库存在，就喝一杯？如果没有搜索到，现在就确认结果。这个笑话的目的不是让大家喝醉，而是想告诉大家，世界上有海量的JavaScript库、框架和插件。同样的观点也适用于单元测试框架。

JavaScript社区提供了许多框架，可以在项目中使用。但是与软件一样，测试框架来去匆匆。在使用之前一定要确认是否处于维护状态。本节会介绍最流行的JavaScript单元测试框架。

388 > QUnit(<http://qunitjs.com/>)是第一个要介绍的单元测试框架，它是使用jQuery开发的，但是是独立的单元测试框架。它已经被jQuery团队管理的其他项目所采用，但也可以用于其他基于JavaScript的代码。QUnit与jQuery 1.x支持的浏览器一样。其中一个好处就是它提供了一套方便的方法来测试项目。此外，除了通用的断言方法，QUnit还能测试异步方法。

Mocha(<http://mochajs.org/>)在Node.js和浏览器中是一个功能丰富的JavaScript测试框架。Mocha测试按顺序执行，允许灵活的和精确的报告，映射未捕获的异常为正确的测试用例。

Jasmine(<http://jasmine.github.io/>)是一个开源的、行为驱动的开发(BDD)框架，它有干净并且易读的语法。

行为驱动的开发

行为驱动的开发是从测试驱动的开发演变而来的软件开发方法。当使用BDD时，不仅可以在单元测试级别测试代码，还可以使用验收测试来测试程序。BDD规定，软件任意单元的测试都应该以期望的行为来指定。

其他大家可能听过的框架，这里要提的就是YUI Test (<http://yuilibrary.com/yui/docs/test/>)和Selenium(<http://docs.seleniumhq.org/>)。框架的使用根据个人喜好、团队技术还有要采用的方法(TDD或BDD)来决定。在本章的剩余部分将会讨论QUnit，因为正如之前提到的，它是jQuery团队开发和维护的框架。如果都相信这个框架，为什么我们不相信呢？下面就从如何使用它开始学习吧。

14.2 QUnit 入门

Getting started with QUnit

QUnit最大的优点之一就是易于使用。入门开发只需要三步即可。

第一步就是下载框架。QUnit可以通过几种方式下载。第一种就是访问官方网站下载最新版的JavaScript和CSS。

注意：编写本书的时候，最新的版本是 1.18.0，但是本章的所有例子都是在 QUnit 2.0 中开发的。

JavaScript 文件包含测试运行器（负责执行测试的代码），以及事件的测试框架（用来测试代码的方法集合）；CSS 文件样式是测试套件的页面，用来显示测试结果。可以在主页面右边找到这些文件的链接，如图 14.1 所示。

第二步是把它们移到一个文件夹里，这里以后也可以创建 HTML 文件。这个文件必须包含引入此 CSS 文件和 JavaScript 文件，还有一个 ID 为 `qunit` 的强制元素标签（通常使用 `<div>`）。在它内部，框架将会创建元素，这个元素用来分组测试和显示结果。配置 HTML 代码的结果如列表 14.1 所示。



图 14.1 QUnit 框架的主页

列表 14.1 QUnit 框架设置的最少代码

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>QUnit Tests</title>
    <link rel="stylesheet" href="qunit-1.18.0.css" />
  </head>
  <body>
    <div id="qunit"></div>
    <script src="qunit-1.18.0.js"></script>
  </body>
</html>
```

包含框架样式表

包装 QUnit 的 UI

包括 QUnit 的 JavaScript 文件

最后一步执行要在浏览器里打开 HTML 文件。一旦打开，就会看到与图 14.2 展示的效果类似的页面。

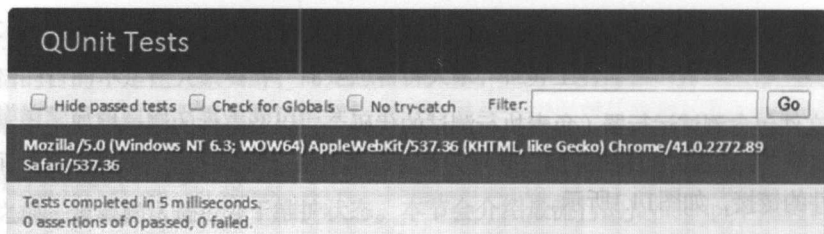


图 14.2 QUnit 框架的用户界面

十分简单，是不是？下面分析这个界面的组件。

页面顶部只有title元素，放置在页面（QUnit Tests）head元素里。在title元素下面，可以看到一个绿条。显示绿条表示所有的测试都通过了（没有测试意味着所有的测试都工作正常）。如果有一个测试失败，则这个绿条会变红。

第二部分可以看到三个复选框：Hide passed tests、Check for Globals（有时候叫noglobals）、No try-catch。当第一个复选框选中时，QUnit会隐藏通过的测试（与期望的结果一致）。第二个复选框允许我们检验在比较之前是否有属性添加到window对象上，在测试失败的情况下，列出差别。当在try-catch之外运行测试代码时，最后一个标志可以用来检验代码会不会抛出异常。除了复选框，还有可以用来过滤搜索测试的input元素。

在第三部分，可以读取window.navigator.userAgent属性。这个属性返回用户浏览器访问网页的用户代理。

底部展示是QUnit测试的时间。下面可以读取定义断言（assertion）的数量、通过的断言数量和失败的断言数量。

什么是断言？

断言是用来检验语句是否等于true的。这对于测试代码是否返回期望的结果十分有用。必须重点注意的是，断言只用来测试有意义的（meaningful）代码。我们的意思是，大家应该只检查自己的代码。例如，你的代码使用了一个原生的JavaScript函数，它无需进行测试。我们必须假定JavaScript函数（例如getElementById()）没有问题，即使有时候会有问题。

正如你所看到的这些数量都是0，原因是还没有定义任何测试代码。没有任何断言，也没有通过和失败的测试。当要编写测试代码和断言时，QUnit将会在这里分组显示断言。

如果在下载文件的时候查看官方的文档，就会注意到它规定了使用测试框架最少的代码。这个元素的ID为qunit-fixture（通常是<div>）。它的目标是阻止测试失败或者避免之前测试的后续副作用，比如删除或者插入元素到DOM里。这个元素有用，因为这个框架会在测试之后重置其中的元素。因此，它不是强制的，但是应该在任意实际的项目中包含它。

其他获取 QUnit 的方法

QUnit也可以通过其他方式获取。第一个可能的方法就是使用jQuery CDN来包含需要的文件。要使用版本1.18.0, 就应该在页面里设置如下代码:

```
<link rel="stylesheet" href="//code.jquery.com/qunit/qunit-1.18.0.css"/>
<scriptsrc="//code.jquery.com/qunit/qunit-1.18.0.js"></script>
```

另外一个方法是使用Bower下载QUnit, 运行下面的命令:

```
bower install --save-dev qunit
```

最后一个方法是使用npm的包管理器来获得:

```
npm install --save-dev qunitjs
```

设置好了代码以后, 就可以开始测试了。但是在这之前, 需要修改一下。首先, 需要在页面中加入要测试的代码或者文件(例如,code.js)。可以把它放到任何地方, 只要在编写的测试代码之前就行。其次就是包含测试的代码和文件。通常放到不同的文件里, 名字叫tests.js。这个文件必须在包含QUnit的JavaScript文件的script元素之后。你的代码应该和下面的差不多:

```
<script src="code.js"></script>
<scriptsrc="qunit-1.18.0.js"></script>
<script src="tests.js"></script>
```

这些文件的内容也可以内联(把内容放置到script元素里)。本章的例子会一直内联测试代码, 而且, 有时要测试的代码也一样放置, 为了简单起见这样做, 但是强烈建议在实际项目的外部文件里使用QUnit。

记住最后一点, 就可以来查看这个框架具体提供的功能了。

14.3 创建同步测试

Creating tests for synchronous code

QUnit允许我们测试同步代码和异步代码。此时, 将会专注于测试同步代码, 因为这是最简单深入学习QUnit的方法。

要创建QUnit测试, 必须调用test()方法, 它的语法如下。

QUnit.test()方法语法

QUnit.test(name, test)

添加要运行的测试。

参数

name(String) 标识创建测试的名称。

392

QUnit.test()方法语法

`test(Function)` 此函数包含运行的断言。框架会传递一个名为`assert`的参数来调用函数。它提供了所有的QUnit断言方法。

返回

未定义。

要创建测试，可编写如下代码。

```
QUnit.test('My first test', function(assert) {  
    //这里编写代码...  
});
```

如果把这个测试放到之前提到的`tests.js`文件里，或者内联放置，然后在浏览器里打开这个HTML文件，就会发现一个错误。框架会抱怨定义了一个没有断言的测试。定义没有任何代码的测试还有什么意义呢？

当创建测试时，最好的方法就是设置期望执行的断言的数量。这可以通过在讨论`QUnit.test()`提到的`assert`参数的`expect()`方法中实现。如果只处理同步代码，那么使用`expect()`方法看起来没有用处，因为认为断言不执行的唯一情况就是出错的时候，是由框架引起的错误。这个观点在处理异步代码的时候是无效的。相信我们，使用`expect()`方法。

`expect()`的语法如下。

`expect()`方法语法

`expect(total)`

设置测试里执行的断言数量。如果实际执行的数量与`total`参数设置的不匹配，测试就会失败。

参数

`total(Number)` 测试要执行的断言的数量。

返回

未定义。

掌握了`expect()`方法的知识后，现在就可以修改之前的代码了，设置期望运行0个测试。

```
QUnit.test('My first test', function(assert) {  
    assert.expect(0);  
});
```

如果重新打开页面，就会发现之前的错误消息消失了。原因是已经显示修改了要执行的断言的数量是0，所以QUnit肯定知道我们想干什么。

通过使用`expect()`方法，我们修复了HTML页面中存在的问题，但是仍然没有设置任何断言。下面看看QUnit中提供的各种不同的断言类型。

14.4 使用断言测试代码

Testing your code using assertions

断言是软件测试的核心，因为它允许我们检验代码是否与期望一致。QUnit提供了许多方法来测试我们的期望，这些方法可以在测试里通过传递给QUnit.test()函数的assert参数访问到。

我们通过介绍四个断言方法来概述断言。

14.4.1 equal()、strictEqual()、notEqual()、notStrictEqual()

本节会介绍QUnit的四个方法。第一个要介绍的是equal()方法，它的语法如下。

equal()方法语法

equal(value, expected[, message])

检验参数value的值是否与expected的值相等，使用不严格的比较(==)。

参数

value(Any) 函数或方法的返回值，或者存在变量里的值。

expected(Any) 要测试的值。

message(String) 断言的可选描述参数。如果忽略，成功时显示消息为“okay”，失败时显示消息为“failed”。

返回

未定义。

断言的介绍里，message的参数是可选的，建议大家一直设置它。

为了让大家理解如何使用这个方法，我们来看一个简单的例子。假设创建了一个求和函数作为参数传递，JavaScript的代码如下：

```
function sum(a, b) {  
    return a + b;  
}
```

sum()函数编写完成以后，要验证它是否工作正常。为此，使用了equal()方法，可以编写如下测试代码：

```
QUnit.test('My first test', function(assert) {  
    assert.expect(3);  
    assert.equal(sum(2, 2), 4, 'Sum of two positive numbers');  
    assert.equal(sum(-2, -2), -4, 'Sum of two negative numbers');  
    assert.equal(sum(2, 0), 2, 'Sum of a positive number and the neutral
```

394

```
    element');  
});
```

可以打开chapter-14/test.1.html文件或者访问JS Bin (<http://jsbin.com/towoxa/edit?html,output>)来运行测试代码。

运行测试验证了代码的正确性和强壮性。但这是完整的测试用例吗？事实证明，不是。如果添加下面的测试断言呢？（记住使用`assert.expect()`更新代码。）

```
assert.equal(sum(-1, true), 0, 'Sum of a negative number and true');
```

这个断言可以成功，因为JavaScript是弱类型语言，所以`true`等价于`1`。因此，求和`-1`与`true`的结果等于`0`。

除了传递Boolean类型的问题，如果传递一个字符串呢？比如`"foo"`。此时函数会把两个参数当成字符串，然后拼接它们。有时候，这种处理方式是期望的，但是通常我们想显示处理这种问题。例如，也许想在参数不是`Number`类型时抛出异常，或者转换数据类型。在讨论异常处理和复杂的情况之前，先来介绍QUnit的另外一个断言方法：`strictEqual()`。

`strictEqual()`方法与`equal()`方法类似，除了它对于实际值和期望值进行严格的比较外。它的语法如下。

strictEqual()方法语法

```
strictEqual(value, expected[, message])
```

使用严格比较符(`===`)检验`value`是否等于`expected`参数。

参数

`value(Any)` 要检验的函数、方法的返回值或者变量。

`expected(Any)` 期望值。

`message(String)` 断言的可选描述参数。如果忽略，成功时显示为“okay”，失败时显示为“failed”。

返回

未定义。

395 为了给`sum()`函数使用`strictEqual()`，现在更新之前的断言代码，可以使用Boolean值`false`来比较两个数字的和(可以更新所有的断言都使用`strictEqual()`)：

```
assert.strictEqual(sum(-2, 2), false, 'Sum of a negative and a positive  
number is equal to false');
```

刷新页面将会显示一个错误。测试能够监测出实际值和期望值不一样。但是使用这种断言（没有什么用），测试失败，不是我们想要的。想法应该是检验两个数的和与Boolean不相等（此时是`false`），而且，如果出现这个结果，则表示断言成功。

对于像断言值不等于或者严格等于另外一个值的情况，可以使用之前介绍的方法的副本：

notEqual()和notStrictEqual()。

更新之前的断言，查看测试是否成功：

```
assert.notStrictEqual(sum(-2, 2), false, 'Sum of a negative and a positive
number is not equal to false');
```

这次刷新HTML页面，将会成功执行四个断言，而且测试成功。

完整版本的页面代码如列表14.2所示。

例子源码可以在chapter-14/test.2.html里找到，也可以在JS Bin(<http://jsbin.com/suroya/edit?html,output>)中找到。

列表 14.2 使用 strictEqual() 和 notStrictEqual()

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>QUnit - Test 2</title>
    <link rel="stylesheet" href="../css/qunit-1.18.0.css" />
  </head>
  <body>
    <div id="qunit"></div>
    <div id="qunit-fixture"></div>
    <script>
      function sum(a, b) {
        return a + b;
      }
    </script>
    <script src="../js/qunit-1.18.0.js"></script>
    <script>
      QUnit.test('My first test', function(assert) {
        assert.expect(4);
        assert.strictEqual(
          sum(2, 2),
          4,
          'Sum of two positive numbers'
        );
        assert.strictEqual(
          sum(-2, -2),
          -4,
          'Sum of two negative numbers'
        );
        assert.strictEqual(
          sum(2, 0),
          2,
          'Sum of a positive number and the neutral element'
        );
        assert.notStrictEqual(
          sum(-2, 2),
          false,
          'Sum of a negative and a positive number is not false'
        );
      });
    </script>
  </body>
</html>
```

QUnit 的 CSS
样式表

测试函数

QUnit 的
JavaScript 文件

sum()函数的
代码测试

```

    });
  });
</script>
</body>
</html>

```

目前为止，只讨论了JavaScript中数值的测试。但是你所学到的方法可以用来测试其他数据类型，比如Array、Object、String等。此外，虽然QUnit可以测试任意的JavaScript代码，但本书主要是关于jQuery的。因此，下面介绍一些使用这些方法来测试用jQuery编写的方法和工具函数的例子代码：

```

assert.equal($.trim(' '), '', 'Trimming a spaces-only string returns an empty string');
assert.strictEqual($('input:checked').length,
  $('input').filter(':checked').length,
  'Filtering elements in advance or later produce the same number of
  elements');
assert.notEqual($('input:checked'), $('input').filter(':checked'),
  'Two jQuery objects are different unless they point to the same memory
  address');
assert.notStrictEqual(new Array(1, 2, 3), [1, 2, 3],
  'Two arrays are different unless they point to the same memory address');

```

正如大家看到的，测试其他不同的数据类型真的与测试Numbers类型没有区别。现在继续学习其他的断言方法。

14.4.2 其他断言方法

QUnit提供的其他断言方法在范围和参数方面十分相似。因此，我们把介绍内容做成如表14.1所示的形式，以方便大家对比学习。

表 14.1 QUnit 提供其他断言方法的概述

方 法	描 述
<code>deepEqual(value, expected[, message])</code>	对所有的 JavaScript 类型进行递归的、严格的比较。断言只有在 value 和 expected 的属性和值相等，并且 value 和 expected 的原型相同时才成功
<code>notDeepEqual(value, expected[, message])</code>	与 <code>deepEqual()</code> 一样，但是用来测试至少有一个属性和值不相等的情况
<code>propEqual(value, expected[, message])</code>	严格比较对象的属性和值。如果所有的属性和值在严格比较时相等，则断言通过
<code>notPropEqual(value, expected[, message])</code>	与 <code>propEqual()</code> 类似，但是用来测试至少有一个属性和值不相等的情况
<code>ok(value[, message])</code>	如果第一个参数是 truthy，则断言通过

实际查看这些方法的使用，构建一个测试数字是否为偶数的函数：

```
function isEven(number) {
  return number % 2 === 0;
}
```

要测试isEven()函数，可以这么写：

```
assert.strictEqual(isEven(4), true, '4 is an even number');
```

但是使用ok()方法可以简化断言代码：

```
assert.ok(isEven(4), '4 is an even number');
```

是不是更好？

deepEqual()和propEqual()的区别很小，但是很重要。要理解这个区别，我们定义一个对象叫human，使用属性fullName，初始化为null。此外，我们定义一个函数叫Person；再定义一个函数叫Person，它用来作为构造器，接受一个字符参数fullName，用来创建对象。函数和对象的代码如下：

```
function Person(fullName) {
  this.fullName = fullName;
}

var human = {
  fullName: null
};
```

现在可以实例化Person类型的对象，设置"John Doe"作为fullName的值，且设置human对象的fullName属性。然后可以使用deepEqual()和propEqual()来测试它们的区别。相关代码可以在chapter-14/test.3.html 和 JS Bin (<http://jsbin.com/tecihe/edit?html,output>)中找到，如下所示：

398

```
QUnit.test('Testing propEqual() and deepEqual()', function(assert) {
  assert.expect(2);

  var person = new Person('John Doe');
  human.fullName = 'John Doe';

  assert.propEqual(person, human, 'Passes. Same properties and values');
  assert.deepEqual(person, human, 'Fails. Same properties and values, but
    different prototype');
});
```

如果运行测试代码，则会看到第一个断言通过，第二个断言失败。原因是person和human有相同的属性和值，但是有不同的原型（prototypes，person有Person作为原型，而human有Object作为原型）。这个条件可以让propEqual()断言通过，但是deepEqual()不行。

下面是最后一个要讨论的断言方法。

14.4.3 throws()断言方法

把throws()断言方法放到最后介绍，因为它与别的方法不同，语法如下。

throws()方法语法

throws(function[, expected][, message])

检验回调函数抛出的异常，可选比较异常值。

参数

function (Function) 要执行的函数。

expected (Object|Function|RegExp) 传递给断言检查的错误对象、错误函数、正则表达式，或者必须返回true的回调函数。

message (String) 断言的可选描述信息。如果忽略，成功时显示为“okay”，失败时显示为“failed”。

返回

未定义。

这个方法与其他方法的不同在于，它接收的第一个参数不是值，而是一个返回异常的函数，期望值是可选的。现在看看实际的代码，让我们修改isEven()函数，使当参数不是Number类型时抛出一个错误：

```
function isEven(number) {  
  if (typeof number !== 'number') {  
    throw new Error('The passed argument is not a number');  
  }  
  return number % 2 === 0;  
}
```

399 要测试抛出的错误，可以这样编写代码：

```
assert.throws(  
  function() {  
    isEven('test');  
  },  
  new Error('The passed argument is not a number'),  
  'Passing a string throws an error'  
);
```

这种情况下，传递的期望值形式与Error一样。另外，可以通过修改之前的断言代码，检验正则表达式的错误信息是不是期望的结果：

```
assert.throws(  
  function() {  
    isEven('test');  
  },  
  /传递的参数不是数值/,  
);
```

```
'Passing a string throws an error'  
);
```

介绍完throws()方法以后,已经学习完QUnit提供的测试同步代码的断言方法。要测试传递给jQuery Ajax回调函数的异步方法,还需要学习一些新方法。现在来看看怎么做。

14.5 如何测试异步任务

How to test asynchronous tasks

有时需要执行一个给定的操作或者重复一定的次数。其他时候想从服务端获取信息而不需要重新加载页面。这些情况都需要执行一个或多个异步操作。

要测试异步函数,可以使用前面学习的创建测试和断言的方法。但是也需要一种机制来通知测试器,我们正在等待异步方法的完成。我们来看下async()方法,它属于前面提到许多次的assert断言参数,它的语法如下:

async()方法语法

async()

通知QUnit来等待异步操作。

参数

无。

返回

唯一的解析回调函数。

400

正如从描述中看到的,这个方法不接受任何参数,并且返回一个函数。这个函数是唯一的,而且必须使用一次,它必须在要测试的异步函数内调用。

为了更好地理解这个方法的工作原理,现在分析列表14.3显示的代码,可以在chapter-14/asynchronous.test.html文件里找到源代码。

列表 14.3 使用 QUnit 测试异步代码

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="utf-8" />  
    <title>QUnit - Asynchronous test</title>  
    <link rel="stylesheet" href="../css/qunit-1.18.0.css" />  
  </head>  
  <body>  
    <div id="qunit"></div>  
    <div id="qunit-fixture"></div>
```

```

<script src="../../js/jquery-1.11.3.min.js"></script>
<script>
    function isEven(number) {
        return number % 2 === 0;
    }
</script>
<script src="../../js/qunit-1.18.0.js"></script>
<script>
    QUnit.test('Testing asynchronous code', function(assert) {
        var $fixtures = $('#qunit-fixture');
        assert.expect(4);

        assert.strictEqual(
            $fixtures.children().length,
            0,
            'The children of qunit-fixture are 0'
        );

        var firstCallback = assert.async();
        window.setTimeout(function() {
            assert.ok(isEven(4), '4 is even');
            firstCallback();
        }, 500);

        var secondCallback = assert.async();
        $fixtures.load('test.1.html #qunit', function() {
            assert.ok(
                true,
                'File test.1.html has been successfully loaded'
            );
            assert.strictEqual(
                $fixtures.children().length,
                1,
                'The elements appended to qunit-fixture are 1'
            );
            secondCallback();
        });
    });
</script>
</body>
</html>

```

① 定义 isEven() 函数
 ② 使用 ID qunit-feature 查询元素
 ③ 使用 async() 创建一个唯一回调函数，存储在 firstCallback 中
 ④ 调用 window.setTimeout() 执行异步函数
 ⑤ 执行存储在 firstCallback 里的函数
 ⑥ 使用 async() 创建第二个唯一回调函数，存储在 secondCallback 里
 ⑦ 使用 load() 异步查询页面
 ⑧ 执行存储在 secondCallback 里的函数

在这段代码里，设置标记使用QUnit，并且包含jQuery。还定义了isEven()函数来测试数字是偶数还是奇数①。然后在包含QUnit测试器之后，可以使用QUnit.test()方法来创建一个异步测试。

在测试的代码里，查询ID为qunit-feature的元素②，后面会向它注入一些元素。然后设置要运行的断言的数量并且创建第一个断言。

接下来调用assert.async()方法创建第一个唯一回调函数，它存储在名为firstCallback的变量③里。第一个要执行的异步操作就是使用window.setTimeout()来延迟500毫秒执行一个函数④。在回调函数内定义，我们测试数字4是不是偶数，而且执行firstCallback⑤中存储

的回调函数。执行这个函数是非常关键的，因为如果忽略它，那么测试器会无限期等待函数，阻止其他的测试执行。

在代码的最后部分，执行`async()`来创建第二个唯一回调函数❹。第二个回调函数存储在`secondCallback`中。其他的异步操作代码使用jQuery的`load()`方法。查询`chapter-14/test.1.html`文件，然后只注入ID为`qunit`的元素❺。在传递给`load()`的回调函数内，使用`assert.ok()`方法来检验文件是否正确加载，而且唯一被注入的元素的ID为`qunit-fixture`。一旦完成，就可以执行存储在变量`secondCallback`里的回调函数❻。

运行异步测试代码十分简单，正如在这个例子中看到的。唯一要记住的就是使用`async()`方法调用回调函数。

有时想使用jQuery Ajax从服务器加载资源。你正在开发的JavaScript代码可能依赖于后台代码，但是后台代码还没有完成，而又不想依赖于后台代码来确保前端代码工作的正常性。对于这种场景，可以使用jQuery Mockjax(<https://github.com/jakerella/jquery-mockjax>)或者Sinon.js(<http://sinonjs.org/>)框架来模拟mock Ajax请求。

本章早期提到了在运行测试时用户界面上QUnit提供的三个复选框。现在讨论其中两个的更多内容，并修改QUnit的行为。

402

14.6 noglobals 与 notrycatch

noglobals and notrycatch

QUnit提供了两个复选框，`noglobals`(标签为“Check for Globals”)和`notrycatch`(标签为“No try-catch”)，通过选择或者取消来控制页面执行测试代码的行为。

如果正在执行的代码引入了一个新的全局变量（与添加到`window`对象的属性一样），`noglobals`将会让测试失败。下面的例子展示了一个如果勾选`noglobals`就会失败的测试：

```
QUnit.test('Testing the noglobals option', function(assert) {
    assert.expect(1);
    window.bookName = 'jQuery in Action';
    assert.strictEqual(bookName, 'jQuery in Action', 'Strings are equal');
});
```

测试失败的原因是`window`对象添加了`bookName`属性。但如果只是修改现有的`window`对象的属性，比如`name`，则测试不会失败。

`notrycatch`标志允许我们运行QUnit，而不需要使用`try-catch`包围。它会阻止QUnit捕获和列出任何错误，但是它允许对问题的深入调试。下面的例子可以在`chapter-14/notrycatch.html`文件里找到源代码，展示了实战的操作：

```
QUnit.test('Testing the notrycatch option', function(assert) {
    assert.expect(1);
```

```
    assert.strictEqual(add(2, 2), 4, 'The sum of 2 plus 2 is equal to 4');  
  });
```

例子代码里的`add()`函数没有定义，所以调用它会导致错误（特别是`ReferenceError`）。如果运行代码激活`notrycatch`标志，则框架不会捕获错误，它会在控制台显示信息。差别如图14.3(a)和14.3(b)所示。

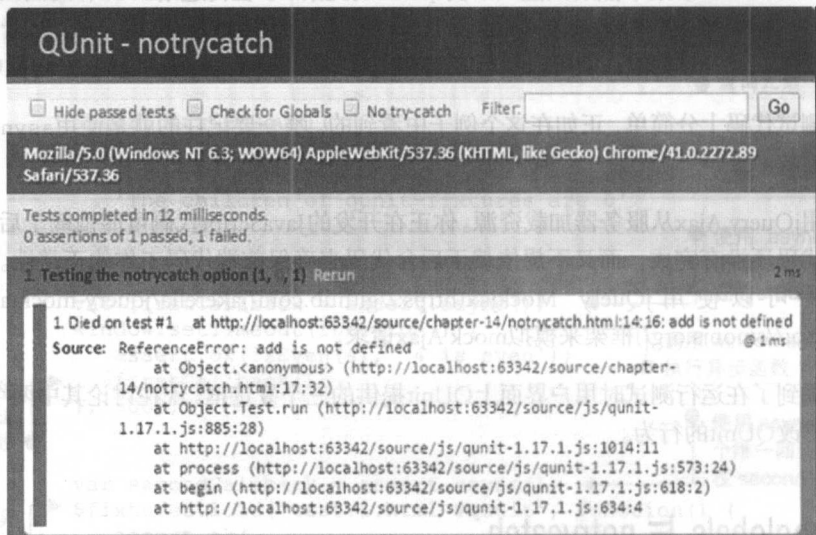


图 14.3(a) 执行没有 `notrycatch` 标志的测试代码

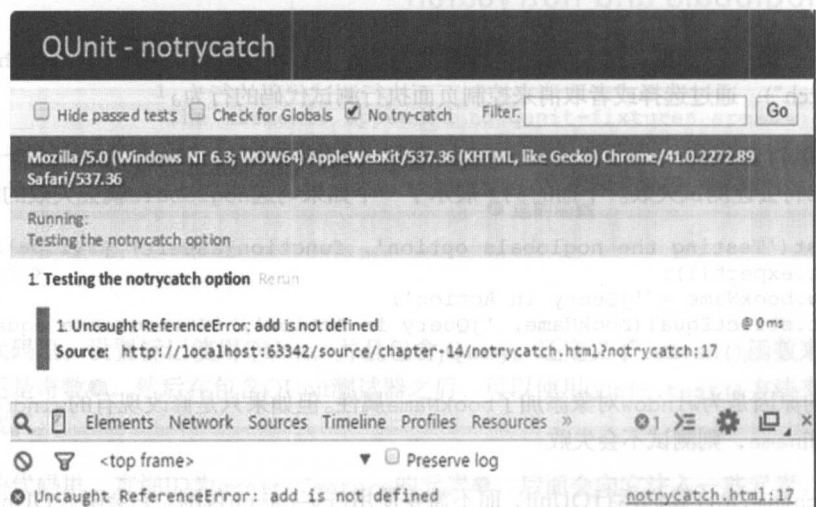


图 14.3(b) 执行没有激活 `notrycatch` 标志的测试代码

当处理大型应用程序时，需要保持测试代码的逻辑组织性，这样就可以运行某些特定组的测试代码而不需要运行全部测试套件，这就是模块出现的原因。

14.7 模块分组测试

Group your tests in modules

随着插件或者库的增加, 你可能希望将源分为模块, 以提高其可维护性。当讨论jQuery结构的时候已经看了这个模式, 它的代码由超过10000行的代码组成。相同的原则也适合我们编写的测试代码。

QUnit提供了简单分组测试代码到模块中的简便方法, 叫`QUnit.module()`。它的语法如下。

QUnit.module()方法语法

QUnit.module(name[, lifecycle])

把相关的测试代码分组到一个模块中。

参数

name(String) 区分模块的名字。

lifecycle(Object) 包含两个选项函数的对象, 在每个测试之前(`beforeEach`)和之后(`afterEach`)运行。每个函数接受一个名为`assert`的参数, 它提供所有的QUnit断言方法。

返回

未定义。

查看这个方法的签名, 也许会猜测如何把测试添加到某个模块中。答案是测试属于某个模块的定义在`QUnit.module()`方法调用之后, 但是在下一次`QUnit.module()`调用之前。下面的代码应该阐明了这个概念。

◀ 404

```
QUnit.module('Core');
```

```
QUnit.test('First test', function(assert) {  
    assert.expect(1);  
    assert.ok(true);  
});
```

```
QUnit.module('Ajax');
```

```
QUnit.test('Second test', function(assert) {  
    assert.expect(1);  
    assert.ok(true);  
});
```

在这段代码中, 加粗字体强调了`QUnit.module()`方法。标记“First test”的测试代码属于Core模块, 标记“Second test”的测试代码属于Ajax模块。

如果实际运行之前的代码, 就会看到测试结果中的模块名前面会有测试名称。此外, 还可以在右上方选择测试运行的模块, 详细细节如图14.4所示。

既然知道了如何用适当的和可维护的方式来组织测试, 现在是时候学习如何设置QUnit框架的配置属性了。

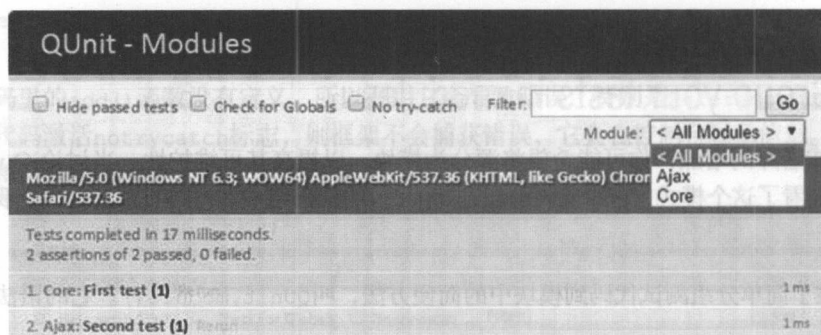


图 14.4 执行模块中的测试代码

14.8 配置 QUnit

Configuring QUnit

与jQuery的许多方法一样，它们包含许多的默认值，QUnit也提供许多预制的配置数据。有时需要修改这些配置来满足特定项目的需求。

框架允许我们重写默认值，通过名为config的属性暴露这些配置。表14.2展示了所有通过405 QUnit.config对象暴露的属性。

表 14.2 QUnit.config 的配置属性

名称	描述
altertitle	(Boolean)Qunit修改 document.title 标题用于添加钩号或者叉号来标记测试结果成功或者失败。如果设置为 false，就会禁用此行为。当代码使用 document.title 的时候，这个设置非常有用
autostart	(Boolean)当 window 对象的 load 事件触发的时候，QUnit 会运行测试。如果异步加载测试，则可以设置为 false(默认为 true)，当所有东西加载完毕后调用 QUnit.start()
hidepassed	(Boolean)QUnit 显示所有的测试，包括传递的。设置此属性为 true，只会看到失败的测试
moduleFilter	(String)指定要运行的测试模块。默认值是 undefined，QUnit 会加载运行所有的模块
reorder	(Boolean)该框架第一个运行之前失败的测试。如果修改此行为，则设置为 false
requireExpects	(Boolean)如果要强制使用 assert.expect() 方法，则设置此属性为 true
testId	(Array)此属性允许 QUnit 通过哈希字符串、模块名、测试名的组合来运行指定的测试块。默认值是 undefined
testTimeout	(Number)设置在所有测试失败之后最长的执行时间。默认值为 undefined，意味着不受限制

名称	描述
scrolltop	(Boolean)如果想要在测试完成后阻止 QUnit 返回页面顶部, 则设置此属性为 false
urlConfig	(Array)控制表单控件放置在 QUnit 工具栏 (toolbar) (可以在里面找到 noglobals 和 notrycatch 标志)。通过扩展这个数组, 可以添加自己的复选框和选择列表

自定义配置文件一定要放到QUnit的JavaScript文件后面。可以在另外的文件里定义配置信息, 代码如下。

```
<script src="qunit-1.18.0.js"></script>
<script src="qunit-config.js"></script>
```

如果测试套件很大, 那么也许我们只关心失败的测试。为此可以设置hidepassed属性, 表14.2介绍了该属性, 代码如下:

```
QUnit.config.hidepassed = true;
```

如果要强制自己或团队指定断言的数量, 则可以使用requireExpects属性:

```
QUnit.config.requireExpects = true;
```

◀ 406

QUnit还定义了一些其他方法, 本章没有介绍, 但它们是次要的。这里介绍的都是重要的知识点, 允许大家针对绝大多数情况来创建自己的测试套件。

第14.9节会实战创建一个完整的测试套件让我们学习QUnit知识。这会很有趣。

14.9 测试套件的例子

An example test suite

在最后一节, 将会为项目实战创建一个完整的测试套件, 还有比测试已开发的代码更好的吗? 这就是为什么会为Jqia Context Menu创建测试套件, 这个在第12章里创建的jQuery插件在页面的某个元素上显示一个自定义菜单。

创建套件的第一步就是为QUnit的基本设置构建一个新页面, 也需要引入jQuery库文件和Jqia Context Menu插件文件(jquery.jqia.contextMenu.css和jquery.jqia.contextMenu.js)。一旦完成, 就有在ID为qunit-fixture的元素里添加一个无序的列表作为自定义菜单, 也会在这个元素上显示自定义菜单。

如果想强制自己和他人使用assert.expect()方法来编写测试套件。为此要修改QUnit的配置requireExpects属性的默认值, 如果没有调用QUnit, 就会抛出异常。

Jqia Context Menu插件组成了一个JavaScript文件, 所以理想情况不需要把测试代码划分为模

块。但是每次必须在测试完成后删除附加给ID为qunit-fixture元素的数据。因此需要使用afterEach，代码如下：

```
QUnit.module('Core', {
  afterEach: function() {
    $('#qunit-fixture').removeData();
  }
});
```

在声明模块之后，就可以定义测试代码了。一共要创建五个测试，分别描述如下。

- **基本需求：**包含检验jQuery和Jqia Context Menu插件是否正确加载的断言。此外，它还检查插件是否配置了正确的默认值。
- **错误参数：**定义检查插件是否可以处理意外参数类型或者缺少强制属性（idMenu）的断言。
- **初始化：**指定断言来检查插件是否破坏链式调用，以及当传递正确参数的时候不抛出错误。此外，它还测试插件是否会在相同的元素上初始化两次或者更多次。
- **回调：**包含测试菜单是否显示或者隐藏的断言，根据元素触发的事件判断。
- **销毁：**测试是否保留链式调用，以及方法是否删除附加给元素的测试数据。此外，它还检查插件特效取消后菜单是否隐藏。

这个测试描述的具体代码如列表14.4所示。

列表 14.4 Jqia Context Menu 的完整测试套件

```
QUnit.test('Basic requirements', function(assert) {
  assert.expect(4);

  assert.ok($, 'jQuery is loaded');
  assert.ok($.fn.jqiaContextMenu, 'The plugin is loaded correctly');
  assert.ok($.fn.jqiaContextMenu.defaults, 'The defaults are exposed');
  assert.propEqual(
    $.fn.jqiaContextMenu.defaults,
    {
      idMenu: null,
      bindLeftClick: false
    },
    'The defaults exposed are correct'
  );
});

QUnit.test('Wrong parameters', function(assert) {
  assert.expect(6);
  var $fixture = $('#qunit-fixture');

  assert.throws(
    function() {
      $fixture.jqiaContextMenu('no method');
    },
    /Method .*? does not exist/,
    'Test plugin can handle wrong parameters'
  );
});
```

创建一个测试来检验是否满足需求

测试插件是否可以处理传递的错误参数

```

    'Call an undefined method'
  );

  assert.throws(
    function() {
      $fixture.jqiaContextMenu(100);
    },
    /Method .*? does not exist/,
    'Wrong argument type: number'
  );

  assert.throws(
    function() {
      $fixture.jqiaContextMenu(null);
    },
    /Method .*? does not exist/,
    'Wrong argument type: null'
  );

  assert.throws(
    function() {
      $fixture.jqiaContextMenu([]);
    },
    /Method .*? does not exist/,
    'Wrong argument type: array'
  );

  assert.throws(
    function() {
      $fixture.jqiaContextMenu({});
    },
    /No menu specified/,
    'Unspecified menu'
  );

  assert.throws(
    function() {
      $fixture.jqiaContextMenu({idMenu: 'unknown id'});
    },
    /The menu specified does not exist/,
    'Unknown menu'
  );
});

QUnit.test('Initialization', function(assert) {
  assert.expect(5);
  var $fixture = $('#qunit-fixture');
  var $fixtureInitialized = $fixture.jqiaContextMenu({
    idMenu: 'context-menu'
  });
  assert.ok($fixtureInitialized, 'Menu initialized');
  assert.notEqual(
    $fixtureInitialized.data('jqiaContextMenu'),
    {},
    'Correct namespace used'
  );
});

```

当传递正确参数时，验证 jqiaContextMenu() 的链式调用性，设置 data-*特性等

```

assert.strictEqual(
    $fixture.length,
    $fixtureInitialized.length,
    'Keep chainability'
);
assert.strictEqual(
    $fixture,
    $fixtureInitialized,
    'Return the same object'
);
assert.throws(
    function() {
        $fixture.jqiaContextMenu({idMenu: 'context-menu'});
    },
    /The plugin has already been initialized/,
    'Plugin already initialized on the element'
);
});

```

测试菜单在点击事件触发后的显示/隐藏功能

```

409 QUnit.test('Callbacks', function(assert) {
    assert.expect(3);
    var $fixture = $('#qunit-fixture').jqiaContextMenu({
        idMenu: 'context-menu'
    });
    var $menu = $('#context-menu');

    assert.strictEqual(
        $menu.css('display'),
        'none',
        'The menu is hidden'
    );
    $fixture.trigger('contextmenu');
    assert.strictEqual(
        $menu.css('display'),
        'block',
        'The menu is displayed after the click'
    );
    $('html').click();
    assert.strictEqual(
        $menu.css('display'),
        'none',
        'The menu is hidden after clicking other elements'
    );
});

```

检验删除 data-*特性后的 destroy()链式调用性

```

QUnit.test('Destroy', function(assert) {
    assert.expect(5);
    var $fixture = $('#qunit-fixture').jqiaContextMenu({
        idMenu: 'context-menu'
    });
    var $fixtureDestroyed = $fixture.jqiaContextMenu('destroy');
    var $menu = $('#context-menu');

    assert.strictEqual(
        $fixture.length,
        $fixtureDestroyed.length,
        'Keep chainability'
    );

```

```

    });
    assert.strictEqual(
      $fixture,
      $fixtureDestroyed,
      'Return the same object'
    );
    assert.strictEqual(
      $menu.css('display'),
      'none',
      'The menu is hidden'
    );
    assert.strictEqual(
      $fixture.data('jqiaContextMenu'),
      undefined,
      'Namespaced data cleared'
    );
    $fixture.trigger('contextmenu');
    assert.strictEqual(
      $menu.css('display'),
      'none',
      'The menu is still hidden after the click'
    );
  });
});

```

410

如果要运行这个测试代码，可以在chapter-14/test.suite.html中找到测试的源代码。

这个测试套件通过了所有的测试，但是为了更好地掌握测试插件的代码，也许要看失败的测试。如果要看一些失败的测试，那么可以按照意外的方式修改插件代码。

如果需要提示建议，那么可以删除return this，可以删除init()和destroy()方法里的代码。这样做会破坏插件的链式调用性，断言会失败。

最后一个demo展示的不仅是本章里介绍的绝大部分测试方法，还有完整的测试套件的样子。希望大家记住这些建议，如果现在还没有为自己的代码做测试，那么可以开始做更多的测试工作了。

14.10 总结

Summary

本章介绍了软件测试的重要概念，以及为什么单元测试非常重要。测试给了我们更多的信心，以确保软件代码工作正常并且没有bug。

也总体介绍了JavaScript单元测试的主流框架，要重点关注QUnit。这个框架也会由jQuery官方团队维护，提供了许多简单、易于使用的测试方法。

在介绍了如何使用QUnit.test()创建测试代码以后，还介绍了几个用来验证返回值的断言方法。也学习了通过assert.expect()方法设置断言数量的重要性。

然后学习了如何测试异步方法，例如，使用`assert.async()`方法测试传递给JavaScript原生函数的`setTimeout()`或者jQuery Ajax的函数。

最后学习了如何使用模块组织测试代码，以及如何为专门的项目设置特定的配置参数。掌握这些知识后，又实战开发了一个完整的测试套件。希望大家从今天或者明天开始测试自己的代码，这样就可以有更强的信心进行重构和使用代码。

411 在本书接下来的最后一章将学习一些有用的工具，可以在大型项目中使用这些工具。

jQuery大型项目开发

How jQuery fits into large projects

本章内容

- 改善选择器性能。
- 在模块中组织代码。
- 使用RequireJS加载代码。
- 使用Bower管理依赖。
- 使用Backbone.js创建SPA。

如果已经阅读了前面所有的章节，那么希望已经掌握了如何使用jQuery编写美观简洁的代码，以及如何扩展它的特性与如何进行单元测试。既然已经知道了jQuery，现在就应该去学习什么时候它不够充分，什么时候使用其他库或者需要框架。

本章是本书的最后一章，我们会扩展几个工具、框架和模式，它们并非与jQuery紧密关联，但是可以用来开发快速、强壮和优美的代码。

jQuery的主要目的就是帮助大家操作DOM。DOM操作通常很慢，所以需要我们明白如何优化jQuery代码来尽快执行操作。还需要了解如何方便地把jQuery集成到大型项目中，以及如何正确地模块化代码来改进代码的可维护性。

< 412

开发人员最大的挑战就是创建高性能的代码。许多人低估了这个问题，但是优化JavaScript代码通常回报丰厚。有时候改进代码的性能非常困难，主要是因为代码太烂只能重构，但是有时候非常简单。第15.1节将会扩展讨论如何通过正确地选择元素来改善jQuery编写的代码性能。

当开发大型项目时，强烈需要良好的代码组织结构。如果代码管理不当，当添加新功能或者重构代码时，就会发现代码乱七八糟，无法扩展。处理这种问题的方法就是使用一些常见的、可靠的模式。组织代码的最佳实践就是使用模块，本章讨论的第二个主题，允许我们更好地概览项目。它也允许我们方便地组织项目，这样不同的开发者就可以开发不同的模块。

分割项目为多个模块，是一种良好的组织代码的方式，但是它也会带来一个新问题。一个模块可能依赖于其他模块去工作，所以要注意页面引用模块的顺序。如果要处理几个模块，这

个问题还容易处理，但是当开发大型项目时，就没有这么简单了。当使用许多模块、插件、库或者框架时，每个都有自己的依赖。其中一个可行的方法是采用RequireJS，这会在第15.3节介绍。

前面提到了插件、库和框架。当开发Web项目时，从0开始编写所有的代码需要耗费大量的时间，所以通常都使用第三方软件代码。最好的例子就是jQuery和Modernizr。要在项目里包含这些组件，访问不同的官方网站，下载需要的文件，然后包含到项目里，放在不同的文件夹即可。虽然这个过程可以成功，但是太慢，而且枯燥乏味。此外，还需要手动检查新的版本。为了自动完成这些复杂的工作，可以使用Bower，会在第15.4节介绍。

最后，在本章的最后一节会介绍Backbone.js。虽然那一节不是这个框架的完整指南，但是可以让大家在以后的学习中有个提前的了解，并且知道如何使用jQuery与其他框架，比如与Backbone.js一起构建大型的复杂应用程序。

15.1 改进选择器性能

Improving the performance of your selectors

当今，高性能是系统开发的重大目标，比以往任何时候都重要。每个Web项目应该从开始避免发布花费10秒钟加载的页面。

高性能代码不仅可以在朋友圈炫耀，还可以提高用户的满意度。本节将学习一些提示和技巧，通过正确使用jQuery选择元素来加速代码。

15.1.1 避免通用选择器

第一个能给的建议，也是最简单的改善选择性能的方法就是避免使用通用选择器(Universal selector)，除非绝对需要。如果代码中有如下的选择器：

```
$('form :checkbox');
```

那么它等价于

```
$('form *:checkbox');
```

正如回忆起来的一样，当在过滤器面前忽略选择器时，通用选择器是隐含假定的。为了改善前面所选择代码的性能，应该这样编写代码：

```
$('form input:checkbox');
```

或者更好的方法，使用context参数，换成下面的代码：

```
$('input:checkbox', 'form');
```

最后一个方法比本节看到的第一个方法大部分情况下都更快。

另外一个应该避免通用选择器的情况是，当查找给定元素的所有子元素时。一种简单的解决方案是使用通用选择器：

```
$('#form > *');
```

但是可以使用更好的代码！更好的方法是使用标签名称选择器，然后使用jQuery的children()函数：

```
$('#form').children();
```

这个解决方案更好，因为它允许jQuery调用JavaScript原生的getElementsByTagName()函数，它非常快。

最后一个例子中展示的原则也可以应用到其他例子中。记住，当jQuery能够调用JavaScript的原生函数时，比如getElementById()（相似的函数中最快）、getElementsByTagName()等，才可以达到最佳的性能。

15.1.2 改进样式选择器

通过本书可以发现，在可能的情况下，jQuery会使用JavaScript原生函数来加速操作执行。

根据样式名选择元素，jQuery库在背后使用了getElementsByClassName()函数：IE9+、Firefox 3+、Chrome、Safari、Opera 9.5+以及其他浏览器。在IE9之前的版本，jQuery仍然可以给我们期望的结果，但是要依赖于自己的实现。因此，如果页面包含大量的元素，那么选择过程的速度可能很慢。 ◀ 414

如果要改进IE6~IE8的性能，例如，如果它们是我们支持的浏览器（某些机构还停留在过去），则可以通过组合样式选择器和元素选择器来优化搜索。特别是，可以为后者加上感兴趣的样式名。

例如，如果想选择包含样式description的p元素，然后存储到变量里，则可以这样编写代码：

```
var $elements = $('#p.description');
```

这是一个好的开端——改善代码的性能，但是还有更多的改进空间。

15.1.3 不要滥用 context 参数

回到第2章介绍的jQuery()的第二个参数，叫context。我们说过，当使用这个参数的时候，它可能通过限制后者到一个或者多个DOM的子树来改善选择的性能，这取决于使用的选择器。

也有一些情况是有context参数而无法改进性能。例如，如果通过ID选择元素，就无法通过指定context参数获得任何好处。而且，这种特定的情况会恶化性能。因此，避免编写下面这样的代码：

```
var $element = $('#test', 'div');
```

因为与下面的代码相比，它会降低性能：

```
var $element = $('#test');
```

第一个方法比较慢，因为库会查询潜在的大量的<div>元素，然后去测试子孙节点，而不是立即使用getElementById()函数。假如大家好奇它有多慢，则可以查看<http://jsperf.com/jquerycontextparameter>的测试结果，如图15.1所示。

图15.1使用jsPerf(<http://jsperf.com>)插件创建，一个允许我们创建和分享测试用例的服务。如果大家不想在自己的机器上运行测试或者分享结果，那么这是一个不错的选择。

感谢这个例子，可以扩展另外一个知识点。为了让jQuery使用getElementById()方法，永远不应该把tag名字预置到ID前面，避免类似\$('p#test')的写法。

使用context可以加速性能的情况通常是当提供ID时。但是，这并非铁律。事实上，当处理性能问题时，没有永远正确或者错误的方法，而是要一种情况一种情况执行测试。

415

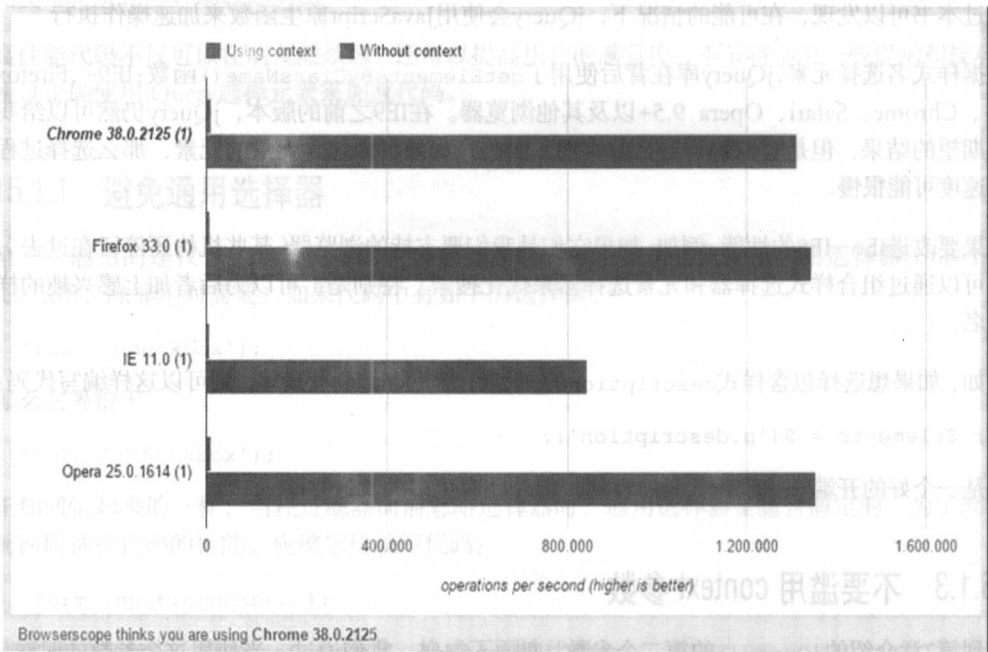


图 15.1 使用 ID 选择元素的性能测试，使用和不使用 context 参数对比性能

性能依赖于许多因素，比如页面元素的数量和类型，以及浏览器。最好的办法就是测试、测

试、再测试，测试选择器来检验特定场景下谁的性能最好。

可能的优化方案还没有讲完。我们再来看看过滤器还有什么优化的空间。

15.1.4 优化过滤器

jQuery支持的许多过滤器并不是CSS规范的内容，所以它们无法使用原生方法带来的性能优势，比如`querySelectorAll()`。比如：`input`、`:visible`等，更好的选择是使用纯CSS选择器，然后使用`filter()`过滤。例如，为了不使用代码：

```
$('#p:visible');
```

可以编写：

```
$('#p').filter(':visible');
```

对于其他过滤器，比如：`image`、`:password`、`:reset`等，可以使用属性过滤器代替。想象一下要查询页面上所有的重置按钮，可以使用：`:reset`过滤器：

```
$('#:reset');
```

但是也可以通过如下代码优化：

```
$('#[type="reset"]');
```

◀ 416

在这个选择里隐式使用了通用选择器，这是我们之前提到应该避免的。为了改进这个选择代码，可以预先考虑元素选择器，代码如下：

```
$('#input[type="reset"]');
```

随着学习的扩展，会看到位置过滤器，特别是：`eq()`、`:lt()`、`:gt()`。与本书讨论的其他过滤器一样，它们都是jQuery扩展的，而不是CSS支持的。为了改进性能，可以先选择元素，然后使用`eq()`方法替换：`eq()`过滤器。这样做，就允许jQuery使用原生的JavaScript方法。作为：`lt()`和：`gt()`替代者，也可以使用`slice()`。

基于这些建议，如果要在列表中选择前两个元素，可以用以下代码：

```
$('#my-list li').slice(0, 2);
```

替代：

```
$('#my-list li:lt(2)');
```

最后一个要提到的优化过滤器是：`:not()`和：`:has()`。在支持`querySelectorAll()`的浏览器里，前者可以通过jQuery的`not()`方法替代，后者可以使用jQuery的`has()`方法替换。

例如，可以把下面的代码：

```
$('#input[placeholder!="Name"]');
```


替换为：

```
$('#input').not('[placeholder="Name"]');
```

介绍完最后一个过滤器优化建议，我们已经完整地介绍了所有可行的优化方法。但是还有一些要分享的独门绝技。

15.1.5 不要过度指定选择器

jQuery依赖于选择器引擎，叫Sizzle，它可以从右到左解析选择器。这意味着为了加速选择，必须右边比左边更准确。为了便于大家理解，假设要在样式revenue的表元素

```
var $values = $('.revenue span.value');
```

而不是：

```
var $values = $('table.revenue .value');
```

由于之前的语句和展示的例子，你可能会过度指定选择器，特别是在右边的。不要这样做！

417 如果可以使用更少的选择器查询相同的元素集合，就去掉无用的部分。

因此，避免如下的代码：

```
var $values = $('table.revenue tr td span.value');
```

可以使用下面的代码选择相同的元素：

```
var $values = $('.revenue span.value');
```

前面的代码难以读懂，而且不会改进性能。

本节提供的建议应该可以帮助大家优化选择器代码。

现在讨论如何通过组织模块来改进项目的结构。

15.2 使用模块组织代码

Organizing your code into modules

当开发大型网站项目时，要格外注意如何恰当地组织代码。限制全局命名空间污染，且提供逻辑化的模块组织方式，都是优先工作，并使用jQuery编写的代码没有异常。应该注意它的代码结构，就像注意其他没有使用jQuery的代码一样。

这个用到JavaScript的简单方法就是在文件中定义几个函数和对象，代码如下：

```
function foo() {};  
function bar() {};  
function baz() {};  
var obj = {};  
var anotherObj = {};
```

这段代码的最大问题是所有的函数和对象都是全局的（window对象的属性可以访问），所以迟早会有一个库加入覆盖这些代码。此外，它不会认为我们需要保留私有数据，这个问题在讨论jQuery插件和jQuery库如何组织的时候已经介绍过。

另外一个问题就是，如果这些函数和对象起着不同的作用，而且被项目的不同部分使用，唯一能够用来辨别这些代码作用的方法就是查看它们的名字。假设要为一个电商网站编写JavaScript代码，而且obj对象和bar()函数都会用来处理支付功能，baz()函数和anotherObj对象是购物车必须的。如何区分它们呢？

使用更多的技术术语，可以说网站有两个模块：支付和购物车。模块在应用程序架构中起着非常重要的作用，正如大家将看到的，可以帮助我们分离和阻止项目的代码。在JavaScript中，有几种方式用于实现模块：AMD (asynchronous module definition,第15.3节讨论)、ECMAScript 2015(也称ECMAScript 6)、对象文本、CommonJS及其他内容。

下面将会学到一些设计模式来组织不同的代码模块。

◀ 418

15.2.1 对象文本模式

其中一个最简单的技巧就是使用对象文本模式来组织代码。为了简单解释这个方法，将会把这个问题分解为几步。

第一步，为了避免污染全局命名空间，就要命名化函数、对象和其他的变量，并使用对象文本模式：

```
var myService = {  
  foo: function() {},  
  bar: function() {},  
  baz: function() {},  
  obj: {},  
  anotherObj: {}  
}
```

有了这个修改，就可以通过单一的入口来访问之前的函数和对象。因此，其他库重写代码的概率就降低了，但是还没有解决根据作用来分离函数和对象的问题：

可以通过进一步扩展前面的方法来解决这个问题：

```
var myService = {  
  foo: function() {},  
  payment: {  
    obj: {},  
    bar: function() {}  
  }  
}
```

定义属于内核的唯一函数

定义支付模块

```

    },
    basket: {
        anotherObj: {},
        baz: function() {}
    }
};

```

← 定义购物车模块

有了这个结构，就可以调用basket模块的baz()函数，代码如下：

```
myService.basket.baz();
```

一旦代码在逻辑上分离完成，就可以把不同的模块放到不同的文件中。可能有一个basket.js文件包含basket模块，定义如下：

```

myService.basket = {
    anotherObj: {},
    baz: function() {}
};

```

由于分离到了不同的文件中，不同的开发者可以更方便开发单独的模块，以减少开发冲突的可能性。

- 419 ▢ 虽然这个方法解决了部分问题，但是它不适合在模块内部保留全局可用的私有数据。换句话说，需要一种方式来创建函数和对象，它们只能在模块内部访问，模块外部无法访问。（在函数内部只能创建函数内部可见的数据）第15.2.2节将会介绍更好的方法来处理这个问题。

15.2.2 模块模式

模块模式已经采用JavaScript概念来模拟（emulate）面向对象编程中的私有方法和内部字段，比如Java或者C#这种面向对象的语言。我们使用模拟（emulate）这个词，因为从技术角度来说，JavaScript里没有private和public这种访问修饰符。

模块模式有两个组成部分：IIFE（附录里介绍的）和返回的对象或者参数。为了让大家快速理解这个主题，下面是此模式的简单代码的实现例子：

```

var myFirstModule = (function() {
    return {
        foo: function() {},
        bar: function () {},
        obj: {}
    }
})();

```

这段代码创建了IIFE，在内部，返回了一个对象文本，它包含要公开暴露的函数和对象。

乍一看，虽然这个模式和前一个模式没有多大区别，但是在本章结束的时候再做定论也不晚。有了这个模式，就可以创建只在模块内部可以访问的变量和函数，外部不可见。假设要添加一个“私有”（private）变量，叫count，用来记录foo()函数调用的次数。还想定义一个“私

有”（private）函数doSomethingPrivate()，无论什么时候执行bar()函数，都会调用它。代码实现如下：

```
var myFirstModule = (function() {  
    var count = 0;  
    function doSomethingPrivate() {};  
  
    return {  
        obj: {},  
        foo: function() { count++; },  
        bar: function() { doSomethingPrivate(); }  
    };  
})();
```

定义“private”变量
定义“private”函数
增加 count 值
调用 doSomethingPrivate()

既然已经明白模块模式的概念，现在介绍其中一个变量，这个变量可以用来为basket模块传递参数：

420

```
window.myService = (function(oldMyService) {  
    oldMyService.basket = {  
        baz: function() {},  
        anotherObj: {}  
    };  
  
    return oldMyService;  
})(window.myService || {});
```

创建 IIFE 并赋值给 window.myService
在 oldMyService 对象上创建 basket 属性，并赋值给它
返回更新的 oldMyService 对象
如果没有定义或传递空对象，则传递 window.myService 作为参数

麻雀虽小，五脏俱全，前面的代码使用了多个技巧。首先，把IIFE返回值赋给window对象上定义的myService属性①。IIFE只有一个定义的参数，叫oldMyService，它会接受window.myService属性的值，或者一个未定义的空的对象（假设值为falsy）②。这样就可以在其他模块（如果已经有了一个）里为myService传递参数或者创建第一个模块。

在IIFE内部，定义了一个名为basket的属性，它所在的对象表示模块，在这个模块里定义了希望暴露的函数和属性③。最后，返回了参数对象④。如果window.myService的值是falsy并且空对象文本传递给IIFE，则这个状态是必须的。

最后一个例子已经简要概括了如何在模块中组织项目代码。虽然还有一些本节中没有介绍的模式，但是现在应该保持代码整洁，并且更加可管理。第15.3节将讨论其他创建模块的模式，叫AMD，并且会介绍RequireJS，它是一个JavaScript模块加载器，它可以加载不同的模块。

15.3 使用 RequireJS 加载模块

Loading modules with RequireJS

第15.2节介绍了两个模块封装代码的方法。但是它们都有一个主要的缺点：必须手动管理每个模块的依赖。例如，某个模块的方法或许需要使用另外一个对象的属性。为了避免这个问

题,要格外注意页面引用这些模块的顺序,但是,对于包含大量模块引用的页面来说,这个工作十分复杂。当某些模块依赖于第三方插件、库或者框架的时候,情况更加糟糕。

解决这个问题的方法之一就是定义模块的依赖,然后封装正确的组织顺序。这也是异步模块定义和RequireJS(<http://requirejs.org/>)诞生的原因。

异步模块定义(AMD)是一个JavaScript API,它定义了一种可以异步加载模块机器依赖的机制。

RequireJS是一个JavaScript和模块加载器,专门为浏览器使用做了优化,但是也可以在其他JavaScript环境中使用,比如Rhino和Node.js。这个库支持高配置,为了更大的灵活性,可以从简单入门开始满足基本的需求。本节虽然不会详细介绍整个库,但是介绍的知识足够大家开始开发了。

RequireJS作为script标签,可在页面的head元素中加载每个依赖。然后,RequireJS会等待所有的加载、所有的依赖,计算模块中函数的正确调用顺序。最后,它会按照正确的顺序来调用模块中定义的函数。

既然了解了RequireJS是什么,以及做什么,现在可以开始使用它了。

15.3.1 开始使用 RequireJS

第一步就是下载代码库。访问页面<http://requirejs.org/docs/download.html>下载最新可用的代码。

在页面中使用RequireJS之前,需要讨论它的几个主要概念。首先要介绍的就是AMD定义的define()函数。

define()方法语法

define([[id,] dependencies,] factory)

使用可选依赖和标识符定义一个新模块。

参数

id(String) 模块的标识符。

dependencies(Array) 包含新模块依赖的模块名称的数组。

factory(Object|Function) 定义新模块的对象文本或者函数。当作为参数提供函数时,它接收依赖的顺序就是定义的顺序。

返回

未定义。

为了强化概念,假设有个对象叫Person,定义在名为Person.js的文件里。它唯一的属性名叫name,而且它没有依赖。使用define()函数可以创建如下代码:

```
define({
  name: 'John Doe'
});
```

正如我们看到的，既没有为这个模块声明ID，也没有依赖。

除了Person，还有一个名为Car的对象，它存储在Car.js文件里。这个对象有一个getOwner()方法，它内部使用了Person对象（文本）的name属性；因此，它对模块Person有依赖关系。

Car模块可以定义的代码如下：

```
define(['Person'], function(Person) {
  function Car() {
    this.getOwner = function() {
      return 'The owner is ' + Person.name;
    };
  }

  return Car;
});
```

这段代码里包含Person作为依赖，然后创建一个名为Car的函数作为构造函数。这个函数包含一个getOwner()方法，它使用Person的name属性返回一个简单消息。最后，作为模块返回Car。目前为止，已经定义了两个模块，但是你仍然没有使用它们。require()函数就是为此存在的。

require()函数与define()的定义类型类似，都定义了模块，但是前者也会执行它。这意味着在执行提供的函数之前，它会加载并执行依赖的模块。通常，应用程序都有一个require()函数作为主入口点，其他模块通过define()定义。

为了完成例子，假设有一个main.js文件，作为应用程序的入口，想在里面弹出汽车所有者的名字。为此，可以使用require()，定义Car作为依赖，代码如下：

```
require(['Car'], function(Car) {
  var car = new Car();
  alert(car.getOwner());
});
```

如果仔细阅读了这一节，脑海中会出现一个问题：RequireJS如何通过一个简单的字符串(例如"Car")来加载模块？答案是库创建了一个同名的模块，作为包含定义的文件。这个例子里，虽然没有为自己的模块定义ID，但是有三个模块名字：main、Car、Person。原因是三个JavaScript文件：main.js、Car.js、Person.js。

下面深入看一下这个约定。如果有一个文件名为Basket.js，存储在名为cart的文件夹里，那么模块将会命名为cart/Basket。

让RequireJS正常工作的最后一步就是，在页面里添加引用。可以使用具备data-main特性的script元素完成。这个特性用来定义应用程序的入口（这个文件使用require()）。假设已经放置好了RequireJS库，而且所有之前创建的模块都在scripts文件夹里，现在可以编写demo了：

423 <script data-main="scripts/main" src="scripts/require.min.js"></script>

包含本节所有代码正常运行的例子可以在chapter-15/requirejs里找到。

既然已经介绍了基本的概念，下面看看如何使用RequireJS和jQuery一起工作。

15.3.2 jQuery 使用 RequireJS

回到第12章，我们介绍了jQuery插件的本质，以及如何自定义扩展jQuery插件。要成为jQuery插件，必须依赖于jQuery。使用jQuery插件编写的代码要依赖于jQuery及其插件。这是一种完美的情形，可以在项目里使用RequireJS。

插件没有从开始开发的时候就使用AMD。因此，看起来只有在每个插件里使用define()，声明jQuery作为依赖。幸运的是，有一个更好的方法，就是第15.4节要讨论的主题。

jQuery 插件使用 AMD

如果从零开始开发插件，而且要使用RequireJS，那么可以通过调用define()来包装插件的定义，声明jQuery作为依赖，代码如下：

```
define(['jquery'], function($) {  
    $.fn.jqia = function() {  
        //这里编写插件代码...  
    };  
});
```

这段代码表示的是不需要返回模块，因为在使用jQuery对象。而且，不需要在IIFE里包装它，因为RequireJS会提供jQuery。前面的代码无法工作，因为它无法解析“jquery”字符串指定的依赖。解决问题的简单办法就是使用RequireJS，把jQuery文件命名为jquery.js，然后放置到相同的目录作为主入口。一旦完成修改，就可以使用我们自己的插件了。

现在假设大家包装的插件代码存储在文件jquery.jqia.js里。假设应用程序主入口存储在main.js里，而且它依赖于插件和jQuery。记住这些需求，main.js代码应该如下：

```
require(['jquery', 'jquery.jqia'], function($) {  
    //使用 jQuery 和 插件的代码  
});
```

这个例子中有两个细节要强调一下。首先，因为代码依赖于插件，后者定义为依赖。其次不需要添加第二个参数来使用插件，因为一旦插件加载完毕，就会调用原始的jQuery对象。

424

使用现有的 jQuery 插件

当开始新项目时，计划采用AMD的架构模块是非常简单的。但是，通常我们要维护旧的库或使用第三方软件，它们没有使用AMD。这种情况下，可以创建一个RequireJS配置文件，以避免修改这些文件来使用define()：

```
requirejs.config({
  shim: {
    'jquery.jqia': ['jquery']
  }
});
```

这个配置使用了shim属性，而且必须放在require()调用前面。shim属性有一个对象作为它的值，允许我们为不调用define()的jQuery插件执行依赖。赋值给shim的对象文本必须定义模块的名称作为其属性，依赖的数组作为其值。

基于这些描述，最后的主文件main.js的代码如下：

```
requirejs.config({
  shim: {
    'jquery.jqia': ['jquery']
  }
});

require(['jquery', 'jquery.jqia'], function($) {
  //使用 jQuery 和 插件的代码
});
```

最后一个例子我们看到了如何在使用jQuery和插件的项目里使用RequireJS。描述的概念远远超过了RequireJS的完整指南，虽然还有很多内容要讨论，像优化器，以及提供的许多配置属性。但是，这是一个好的开端，可以使用这个库来组织项目中的依赖。

同样，需要一个更好的方法来管理模块的依赖，以及在项目中包含的顺序，还需要更快、更好的方法来安装、更新以及删除项目使用的第三方软件。这就是下面将要讨论的内容。

15.4 使用 Bower 管理依赖

Managing dependencies with Bower

Web项目开发通常要使用第三方组件来加速过程。使用一个或者两个第三方组件的项目可以方便进行手动管理，正如几年前的做法一样。随着事情的越来越复杂，开发者需要一个可靠的方法来安装和管理项目的依赖。

过去几年，开发了许多工具来处理这种问题。其中一个工具就是Bower (<http://bower.io/>)。本节将会介绍它的主要特效，关注如何使用Bower把jQuery集成到项目里。

◀ 425

15.4.1 Bower 入门

Bower由Twitter创建，于2012年发布。从此，许多开发者参与到这个项目中，现在它已经是最著名的前端工具之一。Bower定义为Web包管理器，这意味着它是一个JavaScript、CSS等依赖的管理器，比如Web字体库。包可以是JavaScript库（比如jQuery、jQuery UI、QUnit）、

CSS文件(比如 Reset.css 或 Normalize.css)、Web字体(比如 FontAwesome)、框架(比如 Bootstrap)、jQuery插件(比如jQuery Easing 或者 jQuery File Upload)或者其他开发者想使用的任何第三方组件。

有趣的是, Bower本身也有依赖, 所以需要在使用前先满足这些依赖。这些依赖是: Node.js (<http://nodejs.org/>), 这个平台可以允许我们在服务端运行JavaScript语言, 而且本书已经提到过几次; npm (<https://www.npmjs.com/>) 与 Node.js 一起安装; Git 客户端 (<http://git-scm.com/downloads>)。

一旦安装完成, 就可以进入Bower的世界了。

Bower定义了一个清单(manifest)文件叫bower.json, 使用JSON格式编写, 它包含项目的信息, 比如名字、作者和当前版本, 还有使用的包。如果团队开发清单文件非常方便, 则可以与其他成员共享信息。这非常有用, 因为其他开发者也只要输入一个命令就可以安装所有的依赖(后面将会讨论)。

一旦安装了Bower依赖, 就可以通过命令行窗口安装Bower了。

```
npm install -g bower
```

这个过程会持续几分钟, 但是一旦完成, 就可以在项目里使用这个工具了。

假设要开发一个新项目, 而且要使用Bower来管理依赖。开始时, 我们要切换到项目文件夹并在内部创建bower.json文件。这个文件可以手动创建, 也可以使用Bower创建。在这个例子里, 将会讨论后者。打开命令行, 切换到项目文件夹, 运行命令:

```
bower init
```

这个工具将会问你一些关于此项目的问题, 如图15.2所示。

在输入所有字段并确认信息后, 清单文件(bower.json)会在文件夹内部创建。bower.json文件的例子如列表15.1所示。

426

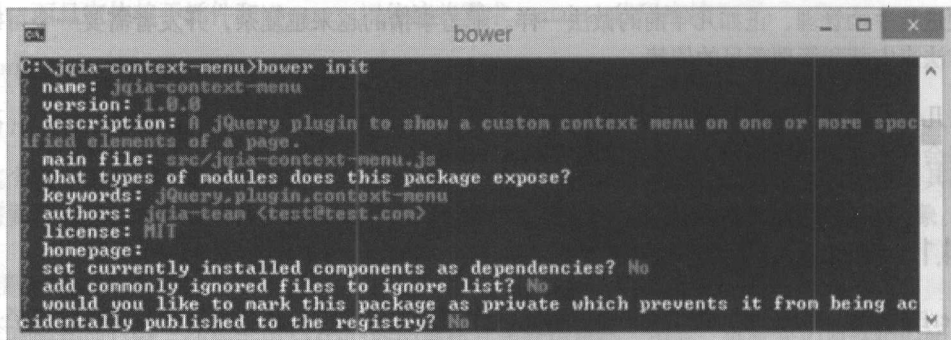


图 15.2 使用 Bower 创建项目清单文件

列表 15.1 bower.json 文件的例子

```
{
  "name": "jqia-context-menu",
  "version": "1.0.0",
  "authors": [
    "jqia-team <test@test.com>"
  ],
  "description": "A jQuery plugin to show a custom context menu on one or
    more specified elements of a page.",
  "main": "src/jqia-context-menu.js",
  "keywords": [
    "jQuery",
    "plugin",
    "context-menu"
  ],
  "license": "MIT"
}
```

这个项目可以设置为使用了Bower，但是目前为止还没有做实际的工作。下面将会继续深入。

15.4.2 搜索包

Bower中的包就是项目中要使用的组件。并非所有的库和框架都能被Bower管理，但是，因为Bower超过了34000个可用包，所以大家可以确信需要的东西基本都可以找到。

如果知道某个包可用，就可以使用Bower。为此，打开CLI运行命令：

```
bower search <package_name>
```

<package_name>表示包的名字。

还有比搜索jQuery更好的例子吗？为了使用Bower搜索jQuery，必须运行命令：

```
bower search jquery
```

427

执行这个命令不仅会搜索到jQuery，还会搜索到名称里包含"jquery"的其他包。

一旦准确找到了使用的包，就可以开始安装了。

15.4.3 安装、更新、删除包

在安装包之前，还有一个重要的决定要做。就是要使用的依赖是必须在生产环境下使用还是在开发阶段使用。

为了方便大家具体理解这个问题，jQuery是一个在生产环境下需要的包，因为整个JavaScript代码，或者部分代码需要依赖于jQuery工作。类似的原因也适合jQuery UI 或者 Backbone.js。其他测试包，例如(QUnit或Mocha)，只在开发阶段用来确保代码的质量和强壮性。该软件的

其他部分——至少这些不会部署——只在开发阶段需要。这是一个非常重要的区别，将会影响我们安装依赖的方式。

使用Bower安装包，要运行以下命令：

```
bower install <package_name>--production-or-development
```

<package_name>是包的名字，<--production-or-development>用来指定是否只在开发阶段使用(--savedev)或者不是(--save)。

为了安装jQuery作为生产环境的依赖，打开命令窗口，切换命令到与bower.json文件所在的项目文件夹。执行命令：

```
bower install jquery --save
```

假设也要安装QUnit，因为要对项目进行单元测试。为此，要安装开发依赖，应输入下面的命令：

```
bower install qunit --save-dev
```

第一次执行install命令，会创建bower_components文件夹。在文件夹内部会下载需要的包，会更新配置文件bower.json的dependency或者devDependency节点，依赖于指定的选项。

一旦依赖下载完毕，比如jQuery，就需要在项目里包含它们。假设有一个index.html文件与bower_components文件夹在一个层次，就必须编写如下代码：

```
<script src="bower_components/jquery/dist/jquery.min.js"></script>
```

实际路径因包而变，但是结构都是类似的。

428 一旦安装完需要的依赖，就可以开发项目的功能了。

开发过程通常需要很长时间，可能包有了新的更新。新版本可能修复了一些重大的bug，所以应该保持依赖更新的及时性。

更新包

更新包非常简单。我们要做的就是将命令行切换到项目根目录，执行CLI命令：

```
bower update <package_name>
```

如果要更新jQuery，可执行如下命令：

```
bower update jquery
```

有时候要一次性更新所有的包。Bower可以执行这种命令：

```
bower update
```

很多时候可能不再需要某个依赖，这时就可以执行删除操作。来看看具体步骤。

删除包

使用Bower可以删除依赖，命令如下：

```
bower uninstall <package_name>[--production-or-development]
```

参数的含义和之前的解释是一样的。

假如在项目里使用QUnit,但是现在不喜欢了,想使用Mocha框架,此时就需要删除QUnit,它可以作为开发依赖进行安装。为此需要执行如下的CLI命令：

```
bower uninstall qunit --save-dev
```

最后一个例子已经完整学习了Bower。这个工具还有更多的命令，但是目前为止介绍的常用命令已经足够满足大家日常项目开发的需求，可加速工作。

幸亏有了jQuery、Bower、RequireJS和其他本书里介绍的工具，可以用更加强壮和专业的方式来开发Web应用了。但是本章还没有介绍单页应用以及如何使用MVC框架开发网站，下面继续扩展学习。

15.5 使用 Backbone.js 创建单页应用

Creating single-page applications with Backbone.js

正如在介绍里讨论的，当开发大型项目时，需要有良好的代码组织结构。想象加入Google公司或者Microsoft公司开发的代码组织混乱是什么结果。对于经常添加新功能的产品或者更新频繁的旧项目，烂代码组织结构会浪费掉大量的时间。众所周知，时间就是金钱。

429

软件架构最常使用的模式之一就是MVC(model-view-controller)架构模式。它把软件架构的关注点分为三个部分：model、view、controller。模型用来表示程序的数据，比如网站用户的注册数据；视图用来显示数据，即网页如何显示数据给用户；控制器用于更新模型的状态，然后发送数据给视图（例如修改用户的地址信息）。

如果你是PHP程序员，则应该知道Symfony、Laravel或者Zend Framework框架；如果你是Java程序员，则应该知道Spring Web MVC 或者Struts；如果你是.NET程序员，则应该知道ASP.NET MVC框架。但是，因为我们在讨论JavaScript，所以有一些不同的框架。其中之一就是实现了MVC模式的Backbone.js。

理想情况下，jQuery可以用来创建单页应用程序，但是这并非它的擅长领域，非要强制实现的话，只能让我们编写出更复杂、难以编写、难以维护的代码。为了创建这种程序，需要一个专门的框架，比如Backbone.js。它有一些依赖库：Underscore.js 和 jQuery。因此，之前学

习的知识也有帮助。

今天,许多Web应用依赖于JavaScript和Ajax来创建更好的用户交互效果和创建单页应用程序(SPA)。这种程序会一次性加载到网页浏览器里,执行部分的HTTP请求,而不需要重新加载整个页面。在性能方面是一大优势,因为无需加载所有的资源(JavaScript文件、CSS文件、字体等)。它只从服务器加载要注入DOM中的一小部分元素。

之前的方法并非没有代价,SPA通常需要更多的加载时间,因为比其他程序需要更多的代码。因此,必须注意平衡代码和应用负载的关系。长时间加载的页面可能带来丢失用户的风险。

下面几节将会讨论这个框架实现模式的几个特性,也会介绍模型、视图和路由的概念。此外,还会开发一个简单的网站来记录工作任务(Todos manager,日程管理器)。我们选择这个程序,是因为它经常拿来作为学习新框架的项目。它的作用就是减少代码量,所以可以轻易记住。请注意本节不打算作为Backbone.js的深入指南,而且会跳过基础部分。如果想深入学习,可以买一本专门的教材来学习。

15.5.1 为什么使用 MV* 框架

430

正如生活中的事物,编程也是一个循序渐进的过程。我们使用自己的工具能解决某个问题,但是当新库和框架发布时,就会完善之前的代码。新的问题出现,然后新的循环开始。

当第一个SPA出现时,许多开发者喜欢用jQuery和Ajax来同步用户输入数据到服务器。一旦应用复杂度增长,代码就会变得无法维护和无法伸缩。后果就是,开发者需要一个专门的框架来帮助他们开发组织良好的和可维护的代码。这也是像Backbone.js、AngularJS、Ember等框架出现的原因。这些框架通常称为MV*框架,因为每个框架都没有完全实现MVC模式、MVP(model-view-presenter)模式和MVVM(model-view-viewmodel)模式。¹

代表MVC的模式以及Backbone.js的实现如图15.3所示。

正如你从图15.3中看到的,Backbone.js与经典MVC模式的主要区别在于视图组件的实现。视图作为一个控制器,用于处理模板模块表示的UI的渲染工作,并负责更新模型。

现在既然已经学习了Backbone.js实现的模型,就可以逐步深入每个概念了,以分析它的职责和特性。

¹ MVC、MVP 和 MVVM 架构模式的区别,译者在高级架构师课程中介绍过多次,这也是企业高级技术职位面试的题目。MVC 于 1974 年由 Trygve Reenskaug 为 Smalltalk 语言创立、解耦模块,代表性框架有 Microsoft 公司的 ASP.NET MVC 和 Java 的 SSH; MVP 于 1996 年由 IBM 公司的 Mike Potel 提出, V 不依赖于 Model, P 代表呈现器,代表框架有 WinForm 和 ASP.NET WebForm; MVVM 于 2005 年由 Microsoft 公司的 WPF 架构师 John Gossman 提出, VM 代表 ViewModel, V 不依赖于 Model, 代表性框架有 WPF 和 Silverlight。—译者注

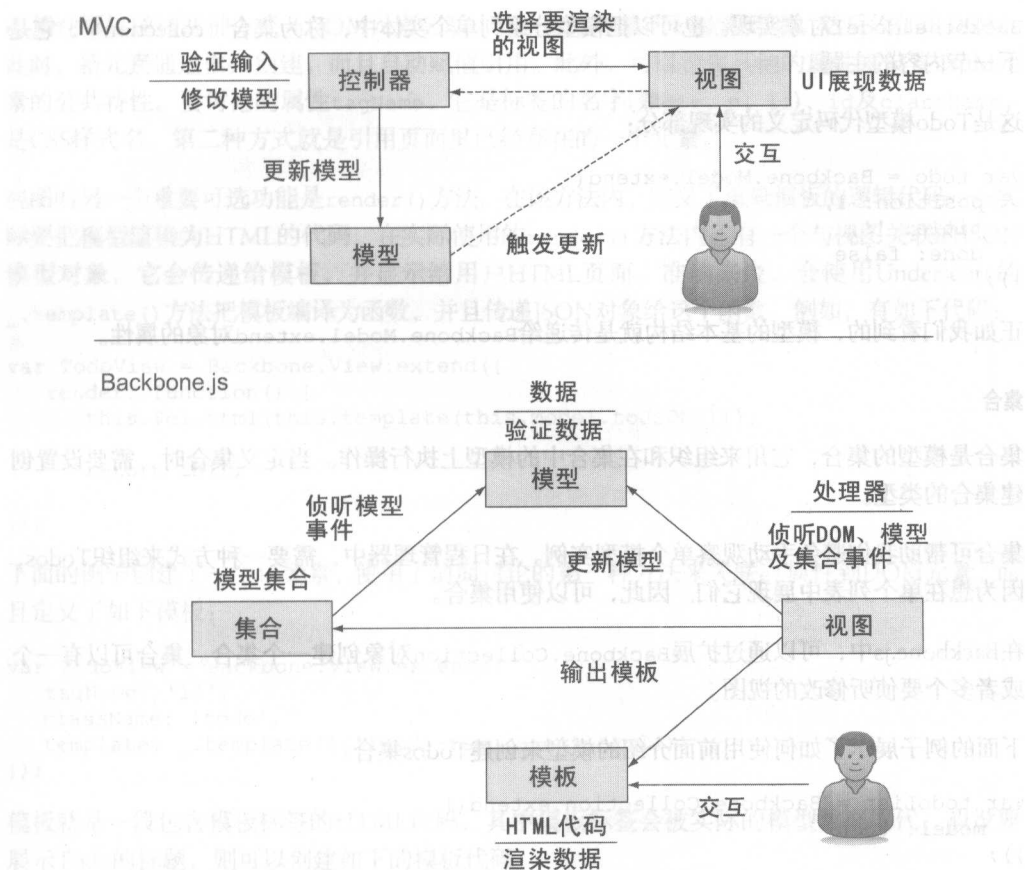


图 15.3 MVC 模式与 Backbone.js 实现的比较

431

15.5.2 开始使用 Backbone.js

正如我们看到的，Backbone.js 允许开发者把代码拆解为更小的模块。

模型

模型是代表应用程序处理的数据。模型具有作为属性的特性，以表示对象的特征。这里通过添加方法来验证数据、初始化属性，而且会通知服务器关于模型的变化。

为了了解模型的意义，请思考日程管理器（Todos manager）。应用程序的主要模型之一就是 Todo 数据输入，它是单个要执行的活动。每个 Todo 对象都包含标题、列表中的位置，以及一个表示其是否完成的属性。这三个对象的属性都属于模型。

模型并不知道如何展示自己包含的数据，而且一个模型可以对应于多个侦听改变的视图。这样，就可以确保视图展示的是模型的最新数据。在 Backbone.js 中，模型通过扩展

Backbone.Model对象实现。也可以把模型分组到单个实体中，称为集合（collection），它是下一节内容的主题。

这是Todo模型代码定义的实现部分：

```
var todo = Backbone.Model.extend({
  position: 1,
  title: '',
  done: false
});
```

正如我们看到的，模型的基本结构就是传递给Backbone.Model.extend对象的属性。

集合

集合是模型的集合，它用来组织和在集合中的模型上执行操作。当定义集合时，需要设置创建集合的类型。

集合可帮助我们避免手动观察单个模型实例。在日程管理器中，需要一种方式来组织Todos，因为想在单个列表中展现它们。因此，可以使用集合。

在Backbone.js中，可以通过扩展Backbone.Collection对象创建一个集合。集合可以有一个或者多个要侦听修改的视图。

432 下面的例子展示了如何使用前面介绍的模型来创建Todos集合：

```
var todoList = Backbone.Collection.extend({
  model: todo
});
```

正如我们看到的，集合可以简单作为只包含模型规定的属性的对象。

视图

视图是通过执行一个或者多个方法来响应DOM事件的组件，通常要绑定特定的模型。视图可以帮助我们保持DOM与数据同步，而且它们是呈现逻辑数据的地方。

这个组件并不包含HTML代码，而HTML代码编写在templates文件里，由专门的JavaScript库Mustache.js 和 Underscore.js管理。Backbone.js依赖于Underscore.js，在demo项目里也会使用到后者。

回忆下例子Todos manager,视图表示的对象允许我们侦听DOM事件，并且运行一个或者多个方法。为了具体理解这个例子，可以考虑Add Todo按钮的click事件或者Todos列表的新模型。

视图最重要的属性是el。它引用了一个所有视图必须包含的DOM元素，而且它把View对象绑定给DOM元素。我们会经常使用jQuery和el上的方法，所以Backbone.js定义了一个方便的\$el属性，它就是jQuery包装el属性的对象。

`el`属性可以通过两种方式与DOM元素关联。第一个是为视图创建新元素，然后添加给DOM。此时，新元素通过框架创建，而且自动赋值引用。此外，可以使用其他的属性来设置DOM元素的公共特性。要讨论的属性`tagName`，它是标签的名字(如`div`、`p`、`li`)、`id`及`className`，是CSS样式名。第二种方式就是引用页面里已经存在的一个元素。

视图的另一个重要可选功能是`render()`方法。在该方法内，定义了渲染模板的逻辑代码——实际要把模型渲染为HTML的代码。在实际使用的`render()`方法内，有一个与视图关联的JSON模型对象，它会传递给模板，并显示给用户HTML页面。准确来说，会使用Underscore的`_.template()`方法把模板编译为函数，并且传递JSON对象给这个函数。例如，有如下代码：

```
var TodoView = Backbone.View.extend({
  render: function() {
    this.$el.html(this.template(this.model.toJSON()));

    return this;
  }
});
```

433

下面的例子创建了一个`li`元素，使用了前面讨论的第一种方法来关联`el`属性到DOM元素，而且定义了如下模板：

```
var TodoView = Backbone.View.extend({
  tagName: 'li',
  className: 'todo',
  template: _.template($('#todo-template').html())
});
```

模板就是一段包含模板标签的HTML代码，其中模板标签会被实际的模型数据取代，假设要展示Todo的标题，则可以创建如下的模板代码：

```
<script type="text/template" id="todo-template">
  <span class="todo-title"><%= title %></span>
</script>
```

综上所述，用户与模板里包含的HTML代码交互被视图管理。视图负责通知模型，而且最终修改HTML代码。

视图还负责传递模型给模板。模板包含要展示数据值的占位符。这些占位符会被实际的模型值所取代。也可以使用其他结构，比如条件语句来判断是否显示某个HTML元素。

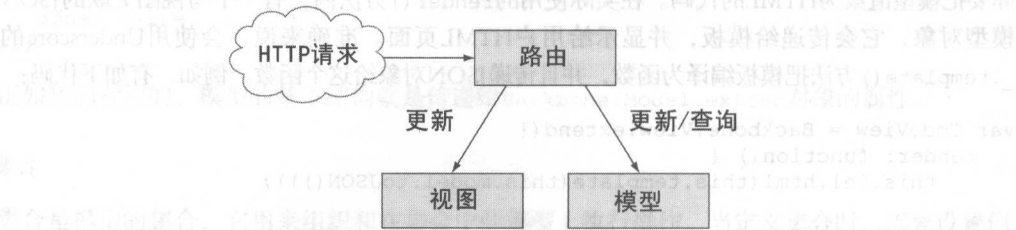
正如模板代码展示的，通过插入自己的内容到`<script>`来创建模板。通常有一个`type="text/template"`属性和一个可以使用jQuery查询的ID。之前引用的模板标签的例子就是`<%= title %>`，`<%= %>`用来获取变量的值，HTML会转义它，`title`是我们期望从视图传输的属性的名字。

路由

路由提供了把项目某个部分与URL绑定的机制，而且保持追踪应用程序的状态。路由可以把

路径映射到函数上，通常与一个或者多个模型一起工作，然后更新视图。图15.4展示了路由如何集成进入Backbone.js架构。

路由可以转换URL或者URL的哈希值为应用程序的状态。这意味着如果需要共享应用程序的状态或者设置标签记录，就需要这些URL。一旦URL匹配既定的路由规则，就会执行对应的函数。



434

图 15.4 路由如何与其他 Backbone.js 组件交互

为了说明问题，我们来回忆之前的Todos manager，想象需要用户来查询特定日程的详细信息，这也是路由的作用。也可以创建路由，并关联到URL，例如格式，todo/MY-TODO-ID——使用函数执行更新页面或者删除Todos列表，且展示特定日程的详细信息。

路由是通过扩展Backbone.Router对象来定义的，而且，虽然可以定义更多的路由，但通常一个应用程序一个路由。路由的定义如列表15.2所示。

列表 15.2 Backbone.js 的简单路由

```
var TodoRouter = Backbone.Router.extend({
  routes: {
    "todo/:id": "getTodo",
    "search/:string": "searchTodo"
  },
  getTodo: function(id) {
    // 代码在这里
  },
  searchTodo(string) {
    // 代码在这里
  }
});
```

❶ 定义路由并映射到函数

❷ 声明路由匹配的函数

在展示路由的例子中定义了两个路由❶todo/:id 和search/:string，并赋值对象给routes属性。对象的键作为路由模式，而且只作为路由匹配时执行的函数(例如，“todo/2”匹配第一个定义的路由)。在这个例子中，todo/:id 和 search/:string会映射到getTodo和searchTodo函数上，它们的函数代码定义如❷部分所示。

正如之前介绍的，路由定义的路径匹配URL，对应的函数被执行，并传递参数变量。变量作为路径的一部分，定义冒号开头的值，比如：id。

解释完这些概念，已经完整分析了框架的组件。现在是时候来开发日程管理器（Todos manager）了。

15.5.3 使用 Backbone.js 创建日程管理器应用程序

当学习新的框架时，大多数人同意通过一个小项目来学习开发是最有效的方式。本节的目标就是带领大家开发一个简单的日程管理器程序（见图15.5），允许用户执行典型的CRUD（增读改删）操作。为了尽可能简单，会使用Web Storage API(<http://www.w3.org/TR/webstorage/>)来发送和存储数据，而不是Web服务，这依赖于Backbone.js 适配器，叫 Backbone.localStorage (<https://github.com/jeromegn/Backbone.localStorage>)。完整的例子代码可以在本书例子源代码chapter-15/todosmanager中找到。

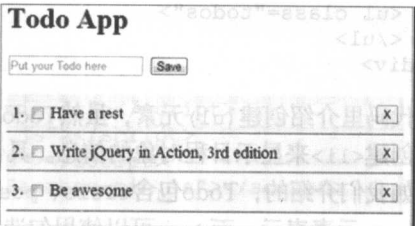


图 15.5 日程管理器应用程序的界面

日程管理器唯一的模型就是Todo，它用来表示要做的事情。每个Todo对象都包含一个title属性，以保存要完成的任务，它的位置通过position属性指定，而且由Boolean值指定任务是否能完成。我们还会使用集合类型来帮助排序。最后可能与大家想到的不同，这里有两个视图。我们将使用元素控制模式（element controller pattern）：第一个控制项目的集合，第二个控制单个项目的实例。

项目结构如图15.6所示，非常简单，包含一个index.html页面，里面是HTML标签和程序使用的模板代码，以及一个包含CSS样式文件的文件夹。JS文件夹包含所有的JavaScript库文件(jQuery和Backbone.js)，在子文件夹vendor内，app.js包含项目代码。为了尽量简单，我们会把所有代码放到一个文件里，但是当出现大型项目时，更好的选择是每个对象都有独立的文件存储在不同的文件夹中：模型、集合和视图。

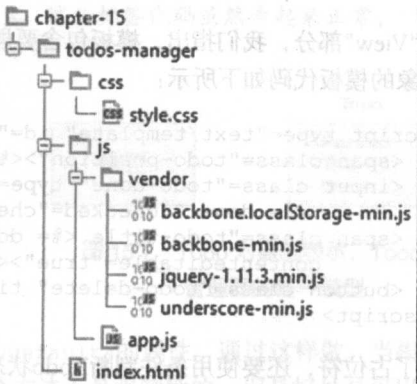


图 15.6 Todo 程序的文件夹和文件结构

既然已经学习了项目的功能，现在可以开始编写代码了。

创建 HTML

没有Web网站不带界面，所以第一步就是开发HTML标签。所有的HTML代码和模板都驻留在index.html文件内。界面很简单，因为只需要两个组件。第一个是用户输入新活动的地方（日程的标题），然后添加到列表；第二个是Todo列表。

开发Web网站时，好的做法是为用户任何失败的操作给出反馈提示。因此，会添加DOM元素来展示必要的错误信息。

这些需求的HTML实现代码如下：

```
<div id="todo-sheet">
  <input id="new-todo" type="text" placeholder="Put your Todo here" />
  <button id="new-todo-save">Save</button>
  <span class="error-message"></span>
  <ul class="todos">
    </ul>
</div>
```

在代码里介绍创建

元素，虽然Todo对象会注入，但是还没有决定如何来显示数据。我们要创建- 来显示日程对象的信息。具体如何做虽然是设计的问题，但是仍然会提出建议。正如我们介绍的，Todo包含title、position、done属性。title和position可以使用简单的span元素表示；而done可以使用勾选框来表示，因为它允许用户来检查，并且表示日程已经完成。

提示：在代码里我们忽略了一个与input字段关联的label元素，为了让大家关注在相关的项目上。然而，当处理表单元素时，最好提供label，因为可以提示用户使用HTML元素。

展示了日程的标题，它可以被编辑，因为设置了contenteditable属性。此外，也可以允许用户使用button按钮删除日程，带有X符号标志。为了给用户足够的反馈提示，你会分配给元素的类，并设置样式的值。

在“View”部分，我们指出，模板包含要展示模型值的占位符，以及一些条件判断语句。Todo对象的模板代码如下所示：

```
<script type="text/template" id="todo-template">
  <span class="todo-position"><%= position %></span>,
  <input class="todo-done" type="checkbox"
    <%= done ? checked="checked" : '' %> title="Completed" />
  <span class="todo-title" <%= done ? 'todo-stroked' : '' %>
    contenteditable="true"><%= title %></span>
  <button class="todo-delete" title="Delete">X</button>
</script>
```

除了占位符，还要使用条件判断Todo状态是否已经完成。

编写完这些代码，需要包含引用项目允许的JavaScript库。

安装 backbone.js

Backbone.js有一个唯一必须依赖的库，叫Underscore.js（版本不低于1.7.0）。它的引用必须在Backbone.js前面；否则，框架无法工作。引用十分简单，所有的工作就像使用<script>添加jQuery引用一样。

项目依赖的库是jQuery, Backbone.js依赖的是Underscore.js和Backbone.localStorageadapter。为了包含这些引用, 只需要在<script>标签里、在index.html页面代码的<body>结束之前引用它们即可, 如列表15.3所示。

列表 15.3 Web 页面中包含库

```
<!DOCTYPE html>
<html>
  <head>
    ...
  </head>
  <body>
    ...
    <script src="js/vendor/jquery-1.11.3.min.js"></script>
    <script src="js/vendor/underscore-min.js"></script>
    <script src="js/vendor/backbone-min.js"></script>
    <script src="js/vendor/backbone.localStorage-min.js"></script>
    <script src="js/app.js"></script>
  </body>
</html>
```

以上代码展示了包含框架是多么简单方便。

注意: 如果要改进这个 demo 项目, 则可以使用 Bower 下载所有的库, 并且使用 RequireJS 管理文件包含的顺序。

为了精益求精, 框架的主页指定了下列信息:

对于 RESTful 持久化, 通过 Backbone.Router 支持, Backbone.View 用来操作 DOM, 包含为了支持旧的 IE 浏览器的 jQuery 和 json2.js。这些标签代码虽然看起来正常, 但是有时候网站无法正常工作。

下面通过开发项目的模型来解决这个问题。

Todo 模型

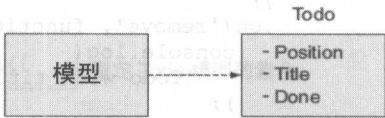


图 15.7 Todo 对象的表示, Todo 管理器的唯一模型

日程管理器唯一的模型就是Todo对象, 它表示要完成的日程活动(见图15.7)。对象的每个实例都具有 title、position、done属性。不是直接添加它们, 如本节“模型”片段所示, 会使用对象来赋值给名为defaults的属性。通过这样做, 当模型创建时, 任意未指定的属性都会设置默认的值。这个方法不是强制性的, 但是好处是可以确保模型的每个属性都设置了默认值。

在模型中创建了两种方法: initialize()和validate()。两者都是可选的, 但是会发现项目中经常使用到它们。前者, 也会使用在集合和视图里, 当创建模型实例时会指定它。现在来添加事件侦听器, 当用户触发事件时可以执行对应的代码。

438

validate()方法默认在存储对象之前调用。这个方法应该在失败时返回错误, 可以是字符串

和对对象，而成功时不返回。当出错时，无论驻留在服务器还是本地，模型不会在存储上更新。这种方法十分重要，因为当返回错误时，它也会触发invalid事件，可以侦听这个事件执行对应的操作。

在项目里，initialize()方法内会侦听这些事件，在控制台上会显示日志信息。通过这种方法可以观察程序正在执行的代码过程。

在深入代码之前，需要执行一个简单的步骤。通过本书，我们已经学习了不污染全局范围的重要性。所有的模型、视图和集合都会包含在app命名空间中。第一行代码如下：

```
window.app = {};
```

设置完命名空间，现在来看看Todo模型的代码，如列表15.4所示。

列表 15.4 Todo 模型

```
app.Todo = Backbone.Model.extend({  
  defaults: {  
    position: 1,  
    title: '',  
    done: false  
  },  
  
  initialize: function() {  
    this  
      .on('invalid', function(model, error) {  
        console.log(error);  
      })  
      .on('add', function(model, error) {  
        console.log(  
          'Todo with title "' + model.get('title') + '" added.'  
        );  
      })  
      .on('remove', function(model, error) {  
        console.log(  
          'Todo with title "' + model.get('title') + '" deleted.'  
        );  
      })  
      .on('change', function(model, error) {  
        console.log(  
          'Todo with title "' + model.get('title') + '" updated.'  
        );  
      });  
  
    validate: function(attributes) {  
      if(!attributes.title) {  
        return 'Title is required and cannot be empty';  
      }  
  
      if(  
        attributes.position === undefined ||  
        parseInt(attributes.position, 10) < 1  
      ) {  
        return 'Position is required and cannot be less than 1';  
      }  
    }  
  }  
});
```

① 扩展 Backbone.Model 对象

② 定义属性的默认值

③ 为无效事件添加处理器

④ 声明验证属性的方法

439

```

        return 'Position must be positive';
    }
}
});

```

在列表15.4的开始通过扩展Backbone.Model创建了一个新对象❶，并且把结果复制给Todo属性，它可以作为属性存储到之前创建的window.app对象中。也可以创建一个对象文本作为特性名字的键，值作为属性的默认值❷。

正如之前提到的，可以创建initialize()方法，也可以为几个事件添加处理器，比如invalid❸。最后，我们重写了validate()方法❹，可以检查title是否为falsy(空string、null、undefined等)，以确保position大于0，并且返回需要的错误信息。

本节构建的模型表示应用程序的待办事项，而且要组成一个Todo集合。下面具体看看。

Todo 集合

想把模型包括到一个单独的列表中。隐藏需要使用Backbone.Collection对象来创建集合，然后设置类型为model的值。也可以强制列表为有序列表，因为可以通过位置来指定每个Todo活动对象。因此，需要定义一个比较器来通过设置集合内的comparator属性，用于排序模型。后者可以是开发人员定义的一种方法，也可以是排序对象属性名的字符串。如果想根据position排序，就需要指定position作为comparator属性的值。

当添加或删除Todo对象时，要保证活动排序的正常位置及位置编号。为此，需要侦听add和remove事件来执行collectionChanged函数，它负责正确排序数据。Backbone.js传递新增或者删除的模型给处理器。这一步很重要，因为你可以测试它是否是有效的数据，使用isValid()方法，而且只有验证通过后才会上更新位置信息。代码实现Todo集合如列表15.5所示。

◀440

列表 15.5 Todo 集合

```

app.todoList = new (Backbone.Collection.extend({
    model: app.Todo,
    localStorage: new Backbone.LocalStorage('todo-list'),
    comparator: 'position',
    initialize: function() {
        this.on('add remove', this.collectionChanged);
    },
    collectionChanged: function(todo) {
        if (todo.isValid()) {
            this.each(function(element, index) {
                element.save({
                    position: index + 1
                });
            });
            this.sort();
        }
    }
}));

```

指定 Todo 模型的集合

定义 Todo 存储的位置

只有模型无效时更新元素

```
});
```

本节讲解了用来存储和分组的Todo数据的对象，但是还没有显示给用户。现在是时候开发视图代码了。

Todo 视图

日程管理器包含两个视图，一个处理单独的Todo活动，另外一个处理Todo活动的集合。本节会讨论前者，后者在下一节中介绍。

Todo日程项目会作为列表显示，单个Todo对象会作为独立项目创建。对于HTML代码，要创建ul和li元素。在li元素内部显示与模型关联的数据：title、position、done。要创建li元素，需要设置view的tagName属性，虽然不是强制的，className属性可以很容易联想到样式的名子。要显示模型的数据，可以使用前面介绍的模板（在“创建HTML”一节里介绍过），并且重写render()方法。

视图也会为单个Todo项目负责响应感兴趣的事件，比如删除或添加单个活动。为此，需要使用Backbone 事件哈希。它就是一个对象，赋值给视图的events属性，由键-值对组成。键由事件名字选择器组成，而且值是要执行的回调函数的名字。

441 执行此视图的代码如列表15.6所示。

列表 15.6 Todo 视图代码

```
app.TodoView = Backbone.View.extend({
  tagName: 'li',
  className: 'todo',

  template: _.template($('#todo-template').html()),

  events: {
    'blur .todo-position': 'updateTodo',
    'change .todo-done': 'updateTodo',
    'keypress .todo-title': 'updateOnEnter',
    'click .todo-delete': 'deleteTodo'
  },

  initialize: function() {
    this.listenTo(this.model, 'change', this.render);
    this.listenTo(this.model, 'destroy', this.remove);
  },

  deleteTodo: function() {
    this.model.destroy();
  },

  updateTodo: function() {
    this.model.save({
      title: $.trim(this.$title.text()),
      position: parseInt(this.$position.text(), 10),
      done: this.$done.is(':checked')
    });
  }
});
```

← 缓存模板

① 事件哈希：关联事件与回调函数

② 从存储里删除模型

← 保存模型

```

    });
  },
  updateOnEnter: function(event) {
    if (event.which === 13) {
      this.updateTodo();
    }
  },
  render: function() {
    this.$el.html(this.template(this.model.toJSON()));
    this.$title = this.$('.todo-title');
    this.$position = this.$('.todo-position');
    this.$done = this.$('.todo-done');

    return this;
  }
});

```

当编辑 Todo 时点击
回车键更新模型

渲染编译
后的模板

列表项目的第一行代码，定义了视图的标签元素，定义了一个元素样式名及模板的缓存。然后创建了事件哈希（events hash）❶，把一系列事件与回调函数关联起来。例如，你想知道，什么时候用户点击Delete按钮从列表来删除模型❷。最后，render()函数用于显示之前编译的模板❸，并且返回包含替换后内容的HTML代码块。

442

现在来讨论第二个应用的视图。

应用程序视图

应用程序的视图称为appView，负责创建新的Todo活动并显示列表。

与Todo视图不同，HTML里已经包含此视图可以引用的DOM元素，所以不需要设置tagName和className。<div>包含todo-sheet的ID，会设置它作为appView属性el的值。当视图初始化的时候，也想获取存储的Todo列表的信息，所以可以显示给用户。为此，可以在initialize()方法的列表上调用fetch()方法。

基于它的职责，唯一感兴趣的DOM事件就是Save按钮的click事件。一旦触发，就可以执行函数createTodo()来创建和存储新的Todo。这种情况完美填充了events哈希表机制。除了DOM事件，视图还需要侦听Todo列表的变化来更新HTML代码显示最新的列表。列表15.7展示了本节讨论的处理代码。

列表 15.7 应用程序视图代码

```

app.appView = Backbone.View.extend({
  el: '#todo-sheet',

  events: {
    'click #new-todo-save': 'createTodo'
  },

  initialize: function() {

```



```

this.$input = this.$('#new-todo');
this.$list = this.$('ul.todos');

this.listenTo(app.todoList, 'reset sort destroy', this.showTodos);
this.listenTo(app.todoList, 'invalid', this.showError);

app.todoList.fetch();

},

createTodo: function() {
  app.todoList.create(
    {
      title: this.$input.val().trim(),
    },
    {
      at: 0,
      validate: true
    }
  );
  this.$input.val('');
},

showError: function(collection, error, model) {
  this
    .$('.error-message')
    .finish()
    .html(error)
    .fadeIn('slow')
    .delay(2000)
    .fadeOut('slow');
},

showTodo: function(todo) {
  if (todo.isValid()) {
    var view = new app.TODOView({ model: todo });
    this.$list.prepend(view.render().el);
  }
},

showTodos: function() {
  this.$list.empty();
  var todos = app.todoList.sortBy(function(element) {
    return -1 * parseInt(element.get('position'), 10);
  });
  for(var i = 0; i < todos.length; i++) {
    this.showTodo(todos[i]);
  }
});

```

① 缓存列表和 input 元素

② 从存储里获取模型

③ 创建新的 Todo

④ 把 Todo 放到列表的顶部，并强制验证

⑤ 显示错误信息给用户

⑥ 如果模型有效，则添加项目

443

`initialize()` 方法里缓存了 `ul` 和 `input` 元素，这里用户可以执行编写新活动的操作(Todo 的 `title`) ①。然后附加对应的事件，最后从本地存储 ② 来获取模型。

在 `createTodo()` 方法里，首先需要创建 Todo 模型的实例，传递标题和属性的默认值 ③。然后

添加到Todo列表的开头,使用at选项,并使用validate强制验证❶。

当发生错误时,通过调用showError()方法在显示样式为error-message的元素里显示信息❷。

要渲染Todo列表,需要创建一个showTodos()方法和一个支持方法showTodo(),用于负责渲染单个Todo信息。在showTodos()方法内,要确保ul列表元素里没有任何内容,可以使用jQuery的empty()方法清空。然后反序列表因为想最后存储元素到列表的头部,显示为第一个列表项。将最后一个元素显示到列表顶部应该怎么做呢?

最后,需要遍历反序的列表调用showTodo()方法,再传递当前的Todo对象。showTodo()方法会检测给定的对象是否有效❸,如果成功,就会创建并展示到ul元素里。

到目前为止,所有的代码都编写完毕,现在可以启动程序了。可以通过编写下面的代码来实现:

```
new app.appView();
```

编写完最后一行代码,整个项目就完成了,现在可以喝一杯啤酒庆祝一下了。最后大家也可以在本书参考代码chapter-15/todos-manager中找到完整的例子。要执行这个程序,只需要在浏览器里打开index.html文件即可。 ◀ 444

本节介绍的Backbone.js只是简要的介绍,而且开发的程序也比较简单。但是我们应该能感觉出来jQuery如何与这个框架紧密集成,以及如何广泛使用在Backbone.js的代码中。这也真正证明jQuery使用的广泛性和灵活性。希望通过这些介绍,大家可以认识到Backbone.js的巨大潜力,并且可以进一步深入学习。最后的挑战是测试我们的知识,希望大家修改项目代码时使用Bower、RequireJS和QUnit。玩得开心!

15.6 总结

Summary

在本章的第一部分,我们介绍了如何正确使用jQuery选择元素来优化性能。讨论了何时使用jQuery()函数的context参数,而且何时避免使用;也讨论了如何避免使用通用选择器(Universal selector)。还介绍了如何通过创建允许jQuery调用原生JavaScript函数,比如getElementById()和getElementsByClassName()的方式来优化旧的浏览器的代码性能。

没有绝对完美的框架。记住,当使用第三方软件,比如强大的jQuery时,还需要做一些优化工作,但是其他的是我们自己的职责。

第15.7节介绍了保持代码干净和良好组织的重要性。告诉大家什么是模块,以及一些使用模块分割代码的可用模式,如何使用jQuery编写(但是不局限于这种情况)代码。其中一个最大的好处就是可以创建私有变量和函数,避免污染全局空间。

另外一个介绍的工具是RequireJS，一个适用于不同环境的JavaScript文件和模块加载器。这个库把我们从手动排序模块、库、框架依赖顺序的复杂工作中解放出来。本节介绍了如何使用RequireJS开发模块，以及如何与现有的代码结合使用。特别是，展示了如何把RequireJS和jQuery插件一起使用，使用简单的配置文件而不需要修改源码。

Bower是我们介绍的另外一个问题。它是Web项目管理JavaScript、CSS以及其他依赖资源的包管理器。本章还介绍了如何使用CLI命令行来查找需要的包，以及如何使用Bower安装、更新和删除文件。

在本章的最后部分学习了Backbone.js，JavaScript中可用的MV*框架之一。Backbone.js允许我们创建单页应用程序(SPA)，一种广泛使用的应用程序类型，可以帮助开发者将后端开发的复杂度降至最低。实际上，绝大部分业务逻辑都是使用JavaScript编写的，而且依赖于客户端。最后一节的Backbone.js内容展示了jQuery框架的扩展使用空间。此外，这个框架也可以和jQuery良好集成，开发出强大的应用。

我们讨论了Backbone.js的架构和主要概念：模块、路由、视图、模板和集合。然后实战开发了一个可以记录日常活动的应用——Todos管理器。

15.7 结尾

The end

天啊！从这本书开始的时候，过去了多久！这真是一段难忘的经历，我们尽力去提供最好的资源，并希望大家达到提升的目标。我们确信这个学习过程对于大家也是难以置信的旅途，而且大家在学习如此大量知识的时候，一定遇到了许多灰心丧气的时刻。如果无法记住我们介绍的每个函数以及函数的参数，不要担心：这没有什么错。经验、实战以及在线查看jQuery官方文档都可以解决这个问题。

jQuery是一个持续发展进化的项目，而且，当阅读jQuery 3文档时会发现它有许多的更新、扩展和弃用，以及删除。有时候我们无法及时了解官方团队的每个库和文档，以及bug查找修复的新闻。

在这本书里，我们努力尝试给大家提供最新版本的jQuery函数和属性的信息、Web开发社区采用的最好的实践开发原则，以及一些高级的编程技巧。

希望大家喜欢，并享受阅读本书的过程，而且不要停止学习，因为学无止境，学海无涯。也祝福大家身体健康、快乐，而且所有的bug都可以被轻而易举地修复！

JavaScript高级编程必备知识

JavaScript that you need to know but might not!

本章内容

- 高效实用jQuery的JavaScript概念。
- JavaScript Object对象基础。
- 函数如何成为一级对象。
- 什么是IIFE?
- 确定（并控制）this的含义。
- 什么是闭包?

jQuery带给Web应用最大的好处之一，就是可以让我们用很少的代码实现强大的功能。jQuery处理了许多底层烦琐的交互工作，可以让开发者专注于实现Web应用的必需功能！

在本书开始的几章里，只需要初级的JavaScript知识就可以理解提供的例子代码。在后面的章节里，比如事件处理、动画和Ajax，必须了解大量JavaScript底层的概念才能有效地使用jQuery库。也许大家已经发现，对之前许多想当然的概念现在有了更深的见解和认识。

◀ 447

我们并非要在本书中阐述所有的JavaScript概念——这并非本书的意图。附录的目的是让大家高效使用jQuery必备的知识。

最重要的概念就是函数是JavaScript中的一等公民，这也是JavaScript定义和使用函数的方式。为了明白函数作为对象的具体意义，首先要理解什么是JavaScript对象。现在就开始深入学习。

A.1 JavaScript 对象基础

JavaScript Object fundamentals

大多数面向对象(OO)语言都定义了一个Object数据类型，其他所有的对象都可以继承自它。

在JavaScript中，Object作为所有对象的基础，但这也是比较的终点。在基础水平上，JavaScript Object与OO语言定义的对象区别很大，共同点很少。

一旦创建，JavaScript Object就不会包含数据，而且语义上公开暴露的数据很少。但是这种语义上的限制也给了它巨大的潜力。下面一探究竟。

A.1.1 如何创建对象

JavaScript有几种创建对象的方式。第一种就是通过new关键字来调用Object的构造函数。这种创建对象的方式十分简单，代码如下：

```
var shinyAndNew = new Object();
```

还有一种更简单的方式（很快就能看到），但是现在先这么做。

新的对象没有内容：没有信息，没有复杂的语义，什么都没有。只有添加东西给它才会有意义——这种东西称为properties（特性）。

A.1.2 对象的特性

JavaScript对象包含数据和方法，它们都可以动态添加到对象中。看下面的例子代码：

```
var ride = new Object();
ride.make = 'Yamaha';
ride.model = 'XT660R';
ride.year = 2014;
ride.purchased = new Date(2015, 4, 10);
```

这里创建了一个Object对象，并赋值给了ride变量。然后给这个变量添加了许多特性（properties）：两个字符串、一个数字、一个日期（Date）实例。

注意：在Date里，月份从0开始。一月对应的顺序是0，二月对应的是1，三月对应的是2，以此类推。

448

不需要在赋值之前声明这些属性，它们只在赋值的时候才会生成。这一点非常强大，给了我们巨大的灵活性。但是通常灵活性都是有代价的！¹

例如，假设后续HTML代码里要修改交易日期的值：

```
ride.purchased = new Date(2015, 7, 21);
```

没有问题……除非打错字了。

¹ 这种动态赋值属性的方式，在其他动态语言或者支持动态语言特性的语言中也存在，Python、Ruby以及C#的Dynamic都支持动态语言特性的实现。——译者注

```
ride.purcahsed = new Date(2015, 7, 21);
```

编译器不会提示错误。purcahsed属性也会成功创建，只是大家会感到疑惑，为什么交易时间没有被修改，直到找到这个打错字的赋值语句的bug。

能力越强，责任越大（大家之前应该听过这话），所以请仔细打字。

从这个例子中，我们学习了JavaScript Object，后面会称为object对象，它是属性的集合。每个属性由名字和值组成。名字可以是字符串，值可以是任意的JavaScript类型：Number、String、Boolean、Object等。这意味着Object实例的主要目的就是作为容器来封装命名的集合数据。

对象属性可以是另外一个新的对象，这个对象也可以有自己的属性，以此类推，嵌套深度取决于要建模的数据类型。

假设现在要为ride添加一个新的属性用来区分不同的汽车车主。这个属性是另外一个JavaScript 对象，包含了多个属性：

```
var owner = new Object();
owner.name = 'Spike Spiegel';
owner.occupation = 'bounty hunter';
ride.owner = owner;
```

要访问嵌套的属性对象，可以编写如下代码：

```
var ownerName = ride.owner.name;
```

嵌套对象的层级没有限制。在结束那一刻，对象的层次结构如图A-1所示。

注意图中的每个值都是不同的JavaScript类型。

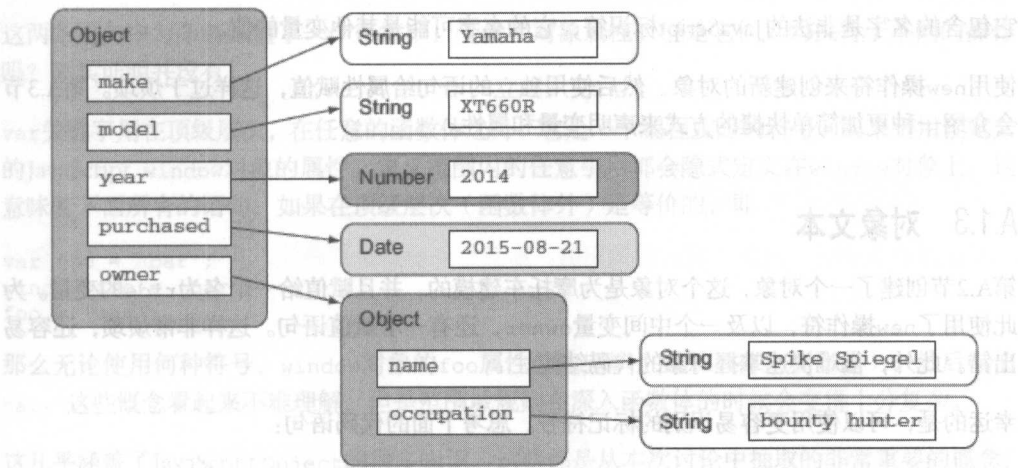


图 A-1 对象层次展示了对象容器引用的其他对象和数据类型的实例

注意：并非所有的情况都需要中间变量（例如 `owner`），这里我们创建了演示代码。后面会看到用更高效的和紧凑的方式来声明对象与属性。

此时，可以通过圆点（`.`）操作符使用对象的属性，这一点的语法与C#的很像。现在思考：假如有一个`color.scheme`属性呢（注意名字中间的圆点操作符）？此时，JavaScript解释器会尝试把`scheme`作为`color`内置的属性。

这并非我们希望的。但是对于空格字符呢？还有其他可能导致误解的分隔符？而且最重要的，如果不知道属性名字，那么可以把它作为其他变量的值或者表达式的计算结果吗？

所有这些情况，圆点操作符都不能满足需求，必须使用方括号操作符（square brackets operator）来访问属性。

```
object[propertyNameExpression]
```

`propertyNameExpression`是JavaScript的表达式，用来计算属性名称。例如，下面所有的引用是等价的：

```
ride.make  
ride['make']  
ride['m' + 'a' + 'k' + 'e']
```

还有这个引用：

```
var p = 'make';  
ride[p];
```

使用方括号操作符是唯一访问名称为非标准JavaScript属性名的属性值的方式，比如：

```
ride["a property name that's rather odd!"]
```

450 它包含的名字是非法的JavaScript标识符，它的名字可能是其他变量的值。

使用`new`操作符来创建新的对象，然后使用独立的语句给属性赋值，这样过于烦琐。第A.3节会介绍一种更加简单快捷的方式来声明变量和属性。

A.1.3 对象文本

第A.2节创建了一个对象，这个对象是为摩托车建模的，并且赋值给一个名为`ride`的变量。为此使用了`new`操作符，以及一个中间变量`owner`，还有一堆赋值语句。这样非常烦琐，还容易出错。此外，很难快速掌握对象的代码结构。

幸运的是，可以使用更容易识别的标记符号。思考下面的代码语句：

```
var ride = {  
  make: 'Yamaha',  
  model: 'XT660R',
```

```
year: 2014,  
purchased: new Date(2015, 7, 21),  
owner: {  
  name: 'Spike Spiegel',  
  occupation: 'bounty hunter'  
}  
};
```

使用对象文本（object literal），这段代码创建了相同的`ride`对象，内置了复制语句，且使用了唯一紧凑的语句。这种方式被绝大部分Web开发者采纳，是比较流行的方式。

这个结构非常简单，对象由大括号组成，里面的属性通过逗号分隔。每个属性的名字和价值通过冒号分隔。正如我们看到的`owner`属性的声明，对象可以嵌套在一起。

也可以使用数组文本符号（array literal notation），它由方括号内的逗号分隔的元素组成，如下所示：

```
var someValues = [2, 3, 5, 7, 11, 13, 17];
```

在本节展示的例子中，对象引用通常存储在变量或者其他对象的属性里。我们来看一下有些特殊的例子。

A.1.4 对象作为窗体属性

到目前为止，我们已经介绍了两种存储JavaScript对象的方式：变量和属性。这两种方式使用了不同的符号，如以下代码所示：

```
var aVariable = 'This is a text.';  
someObject.aProperty = 'This is another text.';
```

◀ 451

这两条语句中每条都赋值了一个字符串：变量和对象属性。但是它们真的执行了不同的操作吗？事实证明并没有！

`var`关键字用在顶级层次，在任意的函数体之外，它是一个编程友好的符号，用来引用预定义的JavaScript `window`对象的属性。定义范围内的任意引用都会隐式定义在`window`对象上。这意味着下面所有的语句，如果在顶级层次（函数体外）是等价的，即

```
var foo = 'bar';  
window.foo = 'bar';  
foo = 'bar';
```

那么无论使用何种符号，`window`对象的`foo`属性创建完毕（如果不存在就创建），然后赋值`bar`。这些概念看起来不难理解，但是范围域规则在深入函数体的时候会变得十分复杂。

这几乎涵盖了JavaScript Object的很多知识。这些都是从本次讨论中抽取的非常重要的概念：

- JavaScript对象是无序的集合属性。

- 属性由名字和值组成。
- 对象可以由对象文本声明。
- 数组可以由数组文本声明。
- window对象的属性属于顶级变量。

现在讨论JavaScript函数作为一级对象的本质意义。

A.2 一等公民函数

Functions as first-class citizens

在许多传统的OO语言里，对象包含数据和方法。这些语言里，数据和方法通常是不同的概念：JavaScript另辟蹊径。

与其他JavaScript的类型一样，函数可以作为对象处理，如String、Number、Date等。与其他对象类似，函数可以通过JavaScript函数定义——此时函数可以

- 赋值给变量；
- 赋值给对象属性；
- 作为参数传递；
- 作为函数结果返回；
- 使用文本创建。

因为在这个语言里，函数与其他对象的处理方式类似，所以我们说函数是一等对象（first-class objects）。

在JavaScript里，函数可以做不同的事情，也可以用不同的方式定义。下面深入进行学习。

A.2.1 函数表达式与函数声明

这看起来有点奇怪，函数不仅可以调用值，而且会证明这是正确的。其中一种定义方式就是函数声明（function declaration）。思考下面的代码：

```
function doSomethingWonderful() {  
    alert('Does something wonderful');  
}
```

函数声明由关键字function组成，接着是函数的名字，还有参数或者无参及函数体。正确的代码里，定义的函数名字是doSomethingWonderful，这是无参的函数。当调用时，就会执行函数体，这里只调用了alert()弹出消息。看起来没有返回值。但是在JavaScript里，如果没

有明确返回值，则默认函数返回的是undefined。

浏览器和函数名

部分或者全部支持ECMAScript 6规范的浏览器都会保留一个name属性来存储函数名。可以在https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/name中学习到的更多内容。

前面提到过顶级定义的变量会作为window对象的属性。Function对象也不例外。如果之前的函数声明在顶级层次，那么可以创建与函数名同名的window属性。因此，下面的语句是等价的：

```
function hello() { alert('Hi there!'); }
hello = function hello() { alert('Hi there!'); }
window.hello = function hello() { alert('Hi there!'); }
```

在部分支持或者全部支持ECMAScript 6规范的浏览器里，可以通过名为name的函数属性名访问函数。

在JavaScript里，函数可以作为代码的一部分定义，而且被称为函数表达式（function expression）。函数表达式的值可以作为函数对象。考虑下面的代码：

```
var myFunc = function() {
    alert('this is a function');
};
```

453

正如我们看到的，定义了一个变量myFunc，赋值了一个函数。因为这是一条代码，注意这个函数没有名字（它的name属性是空字符串），所以不能使用函数名调用它。但是，因为已经赋值给一个变量，所以可以按如下方式来执行：

```
myFunc();
```

这并非函数声明与函数表达式的唯一区别。另外一个重要的区别：函数声明是升起的（hoisted），而函数表达式不是。为了实际理解这个概念，思考下面的例子：

```
funcDecl();
funcExpr();

function funcDecl() {
    alert('function declaration');
}

var funcExpr = function() {
    alert('function expression');
};
```

消息正确弹出

错误！

通过函数声明创建函数

通过函数表达式创建函数

例子中定义了两个函数funcDecl() ❶和funcExpr() ❷。但是在实际定义之前，我们执行了调用。首先调用（funcDecl();）成功，然后调用(funcExpr();)抛出错误。不同的行为是因为

`funcDecl()` 函数是升起的, 而 `funcExpr()` 不是。

可以采用同样的方式为变量赋值函数表达式, 也可以作为属性赋值给对象:

```
var myObj = {  
  bar: function() {}  
};
```

我们已经看过给变量和属性赋值函数的例子, 那么把函数作为参数传递如何? 下面看看为什么以及怎么实现。

A.2.2 回调函数

当处理事件或计时器, 或者执行Ajax请求时, Web页面代码的本性是异步的。其中异步编程最流行的一个概念就是回调函数。

下面看看计时器的例子。可以调用计时器来触发——假设5秒钟——通过传递适当的间隔时间给 `window.setTimeout()` 方法。但是这个方法怎么让等待时间结束后执行想要的方法? 也是通过调用设置的函数实现的。

454

思考下面的代码:

```
function hello() { alert('Hi there!'); }  
setTimeout(hello, 5000);
```

我们定义了一个名为 `hello` 的函数, 设置计时器5秒后触发, 通过第二个参数5000毫秒控制触发时间。第一个参数设置给 `setTimeout()` 方法, 是函数的引用。传递函数作为参数与传递其他值没有区别, 就好像传递一个数一样。

当计时器结束时, 则调用 `hello` 函数。因为 `setTimeout()` 方法在代码里发起了一个回调, 所以函数被称为回调函数 (callback function)。

绝大部分高级JavaScript工程师应该觉得这个例子的代码简单, 因为没有必要创建函数名 `hello`, 只使用一次函数。除非要在其他地方多次调用函数, 否则没有必要创建 `window` 属性来存储 `hello` 函数并把它传递给回调参数。

更加优美的代码方式如下:

```
setTimeout(function() { alert('Hi there!'); }, 5000);
```

这里在参数列表里直接定义 (实际是内联匿名函数), 无须生成函数名字。可在jQuery中经常看到这种用法, 没有必要赋值给顶级属性。

在这个例子里创建的函数或者是顶级函数 (作为 `window` 属性), 或者赋值给函数调用。也可以复制 `Function` 对象给对象的属性。下面来一探究竟。

A.2.3 寻根求源

OO语言会自动提供一种方式来引用当前方法内对象的实例。在Java和C#这种语言中，`this`变量指向当前实例的引用。在JavaScript中，类似的概念存在，也使用`this`关键字，还可以访问与函数关联的对象。但是JavaScript实现的`this`与OO语言的不同。

在OO语言中，`this`通常引用的是声明方法类的实例。在JavaScript中，函数是一等对象，不会作为其他东西的一部分，对于通过`this`引用——称为函数上下文（`function context`）——不是由函数声明来决定而是由函数调用（`invoked`）来确定的。

这意味着相同的函数可以有不同的上下文依赖，取决于如何调用它。这一点看起来有些诡异，但是非常有用。

默认情况下，函数调用的上下文(`this`)属性包含了对于调用函数引用的对象。我们来看看摩托车代码的例子，修改对象的创建代码如下（新加的代码用粗体显示）：

455

```
var ride = {  
  make: 'Yamaha',  
  model: 'XT660R',  
  year: 2014,  
  purchased: new Date(2015, 7, 21),  
  owner: {  
    name: 'Spike Spiegel',  
    occupation: 'bounty hunter'  
  },  
  whatAmI: function() {  
    return this.year + ' ' + this.make + ' ' + this.model;  
  }  
};
```

新代码添加了一个名为`whatAmI`的属性，它引用了一个函数`Function`实例。新的对象层次，使用`Function`实例赋值给一个名为`whatAmI`的属性，如图A.2所示。

当函数通过这个属性引用调用时，代码如下：

```
var bike = ride.whatAmI();
```

函数上下文(`this`)设置为`ride`引用的对象实例。结果变量`bike`设置为'`2014 Yamaha XT660R`'，因为函数通过`this`获得了对象的属性。

456

对于顶级函数也一样。记住顶级函数是`window`对象的属性，所以调用函数上下文是`window`对象。

虽然是通常和隐含的行为，但是JavaScript给了我们现实控制函数上下文的机会。可以通过`Function`方法`call()`或者`apply()`来设置调用函数的上下文。虽然看起来有点疯狂，但是作为一等对象，函数有通过`Function`构造函数定义的方法。

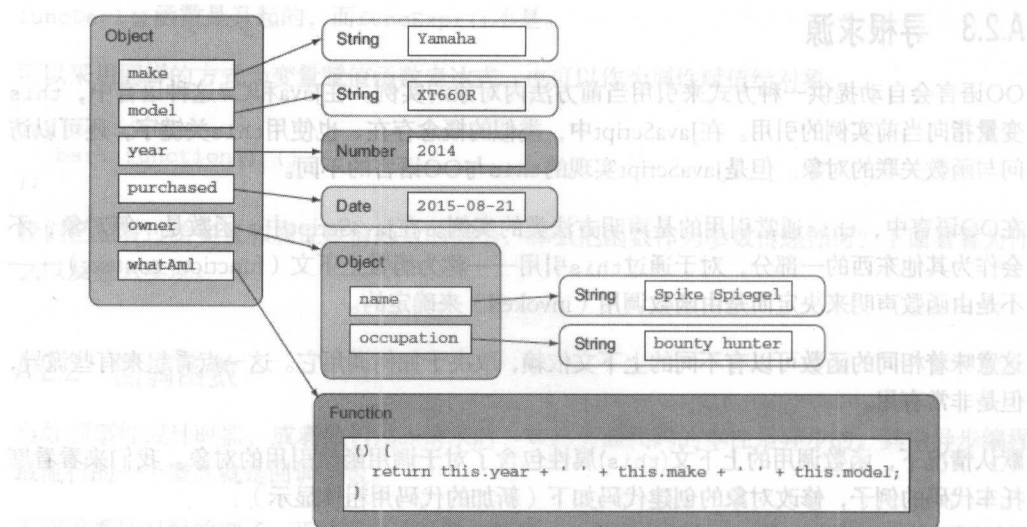


图 A.2 模型清晰展示了函数不是对象的一部分，但是可以通过对象的属性 `whatAml` 访问引用 `call()` 方法调用函数指定第一个对象参数作为上下文，而剩余的参数作为调用函数使用——`call()` 的第二个参数变成调用函数的第一个参数，以此类推。`apply()` 方法与此方法的工作方式类似，除了第二个参数是数组参数用来调用函数使用。

为了强化概念，我们来看一个例子，思考列表A.1中的代码(可以在本书例子代码appendix-a/function.context.html和JS Bin at <http://jsbin.com/dumac/edit?html,js,output>中下载)。

列表 A.1 函数上下文的值取决于函数调用

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Function Context Example</title>
  </head>
  <body>
    <script>
      var obj1 = { handle: 'obj1' };
      var obj2 = { handle: 'obj2' };
      var obj3 = { handle: 'obj3' };
      var value = 'test';
      window.handle = 'window';

      function whoAmI(param) {
        return this.handle + ' ' + param;
      }

      obj1.identifyMe = whoAmI;
    </script>
  </body>
</html>

```

① 使用相同的属性名定义三个不同的对象，设置不同值

② 使用一个参数定义函数

③ 把函数赋值给obj1对象的属性

```

调用 whoAmI()函数 ④
<script>
    alert(whoAmI(value));
    alert(obj1.identifyMe(value));
    alert(whoAmI.call(obj2, value));
    alert(whoAmI.apply(obj3, [value]));
</script>
</body>
</html>
使用 apply()调用 whoAmI()函数 ⑦
使用 call()调用 whoAmI()函数 ⑥
使用存储在 obj1.identifyMe 引用调用 whoAmI()函数 ⑤

```

457

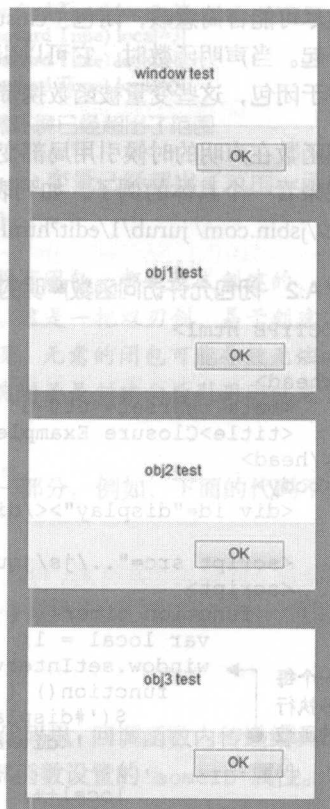
代码里定义了三个对象，每个对象使用handle属性来区分对象的引用④。同样也为handle实例添加了属性，因此它易于辨认。

然后定义了一个顶级函数，它可以返回任意作为任意函数上下文对象的handle属性的值②，并把同一个函数赋值给obj1的identifyMe属性③。可以说在obj1上创建了一个名为identifyMe的方法，虽然函数是和对象独立声明的。

最后弹出四个警告框，每个警告框使用一种不同的机制来调用相同的函数实例。当加载到浏览器时，四个警告框的顺序如图A.3所示。

警告框的顺序如下：

- 当函数直接作为顶级函数调用时，函数上下文是window实例④。
- 当作为对象属性 (obj1)时，对象就变为函数的上下文⑤。可以说函数作为对象的方法来运行——像OO语言一样。但是不要过于乐观，否则会走火入魔，过度解读，因为这个例子的结果会显示出来。
- 使用函数Function的call()方法，会导致函数上下文传递给call()的第一个参数对象作为上下文——这个例子中的obj2⑥。函数看起来像obj2的方法，尽管没有什么关系——作为属性——使用obj2。它展示了当使用call()时如何传递参数。
- 与call()一样，Function的apply()方法会设置函数的上下文为传递的第一个参数对象⑦。两种方式区别很大，尤其在传参的时候。实际上，当使用apply()方法时，所有的参数都必须作为单个数组传递，且数组参数作为第二个参数。



图A.3 作为函数调用上下文的对象改变了函数调用的方式

458

图A.3这个页面清晰演示了函数上下文取决于调用的环境，而且单个函数可以使用任意不同的对象作为上下文调用。因此，可能无法说函数是对象的方法，所

以必须要澄清一下：

当obj作为函数func调用的上下文时，函数func作为对象obj的方法。

为了更进一步演示这个概念，思考下面的代码，并将其添加到例子中：

```
alert(obj1.identifyMe.call(obj3));
```

虽然作为obj1的属性引用了函数，但这时调用函数上下文是obj3，也进一步强调了函数声明无法决定上下文而是取决于如何调用函数。

既然已经理解了函数如何作为对象方法，那么现在把注意力转移到另外一个高效jQuery编程的知识点：闭包。

A.2.4 闭包

表述尽可能言简意赅，闭包（closure）指的是Function实例，与其执行需要的局部变量耦合在一起。当声明函数时，它可以引用自己范围内的任意变量。这一点大家应该十分清楚。但是对于闭包，这些变量被函数携带，甚至在声明点之后，已经超出了范围，关闭声明。

回调函数在声明的时候引用局部变量是一个编写高效JavaScript代码的必备工具。使用计时器，我们来看一个具体的例子，如列表A.2所示。例子代码可以在appendix-a/closure.html文件和<http://jsbin.com/jurub/1/edit?html,js,output>里找到。

列表 A.2 闭包允许访问函数声明的范围

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Closure Example</title>
  </head>
  <body>
    <div id="display"></div>

    <script src="../../js/jquery-1.11.1.min.js"></script>
    <script>
      function timer() {
        var local = 1;
        window.setInterval(
          function() {
            $('#display').append(
              '<div>At ' + new Date() + ' local=' + local + '</div>'
            );
            local++;
          },
          2000
        );
      }
    </script>
  </body>
</html>
```

定义一个每隔 2 秒执行的函数 ①

① 定义当前时间写入的元素

② 初始化局部变量为 1

④ 增加局部变量

```

timer();
</script>
</body>
</html>

```

执行 timer() 函数

在这个例子中，我们创建了一个函数 `timer()`，在定义之后执行⑤。在 `timer()` 函数内声明了局部变量 `local`②，并且赋值为1。然后使用 `window.setInterval()` 方法来建立计时器，每隔2秒触发一次③。作为计时器的回调函数，我们指定了一个内联函数来引用局部变量 `local`，通过向页面里名为 `display` 的元素附加 `div` 来展示当前时间和 `local` 变量的值①。作为回调函数的一部分，`local` 变量的值每次递增1④。

如果不了解闭包，那么也许会认为，因为回调函数会在 `timer()` 函数调用后2秒触发，`local` 变量的值在执行回调期间是未定义的。但是，加载页面并运行一小段时间，会看到如图A.4所示的结果。

```

At Sun Mar 15 2015 01:12:21 GMT+0000 (GMT Standard Time) local=1
At Sun Mar 15 2015 01:12:23 GMT+0000 (GMT Standard Time) local=2
At Sun Mar 15 2015 01:12:25 GMT+0000 (GMT Standard Time) local=3
At Sun Mar 15 2015 01:12:27 GMT+0000 (GMT Standard Time) local=4
At Sun Mar 15 2015 01:12:29 GMT+0000 (GMT Standard Time) local=5

```

图 A.4 闭包允许回调函数访问自己的环境，虽然环境已经超出了范围

这个例子可以正常工作，虽然当 `ready` 处理器已经退出，`local` 变量已经超出了范围，函数声明的闭包，包含 `local`，仍然存在于函数的生命周期范围内。

注意：你可能也注意到了闭包，对于 JavaScript 中的所有闭包，都是隐式创建的，并不需要显式的语言，正如其他支持闭包的语言一样。这是一把双刃剑，易于创建闭包（无论你想或者不想！），但是也难以在代码中发现。无意的闭包可能导致无法预测的结果。例如，循环引用可能导致内存泄露。经典例子是创建向后引用闭包变量的 DOM 元素，阻止这些变量被回收。

460

闭包的另外一个特性是，函数上下文从来不会作为闭包的一部分。例如，下面的代码不会按照我们的预期执行。

```

...
this.id = 'someID';
$('*).each(function() {
    alert(this.id);
});

```

记住每个函数调用都有自己的函数上下文，所以，在前面的代码里，回调函数内传递给 `each()` 函数上下文是 jQuery 集合中的元素（DOM 元素），不是外部函数设置的 `'someID'` 属性。每次调用回调函数都会轮流显示 jQuery 集合中的每个元素的 ID。

当访问作为函数上下文的对象时，可以在局部变量里使用常见的版本来创建 `this` 引用的拷贝，这个局部变量将会包括到闭包里。思考下面的修改代码：

```

this.id = 'someID';
var outer = this;
$('*').each(function() {
    alert(outer.id);
});

```

变量outer（绝大部分时间命名为that）变成了闭包的一部分，因为它已经在回调函数内部被引用了，因此可以被访问。outer赋值给任意上下文，而不是回调函数定义的上下文。例如，前面的代码包括在名为foo的函数内，outer变量会引用foo函数的上下文。如果前面的代码定义在HTML页面里，而没有被函数包裹，则outer变量将会引用window对象。

修改后的代码现在显示警告框来展示字符串'someID'任意多次，只要jQuery集合中有元素就行。

我们发现闭包在使用jQuery异步回调来创建优美代码时非常重要，尤其是在编写Ajax请求和事件处理代码时。

在结束附录内容之前，想和大家讨论最后一个概念：立即调用的函数表达式IIFE。

A.2.5 立即调用的函数表达式 IIFE

IIFE是一种JavaScript设计模式，它用来创建函数表达式，然后立即执行函数。由Ben Alman在他的博客首次提出“immediately-invoked function expression (IIFE)”(<http://benalman.com/news/2010/11/immediately-invoked-function-expression/>)。

在讨论使用IIFE的好处之前，让我们看看如何实现管理它。

```

(function() {
    //函数的实现代码...
})();

```

在这段代码里，我们创建了一个匿名函数，由于最后的()，故它可以立即执行。函数被包装在两个括号里，因为想告诉解析器这是一个函数表达式而不是函数声明。

几种场景下此模式非常有用。可以使用它来创建私有变量以及函数范围外部无法访问的函数。其他的好处是因为可以访问私有的变量和函数，就可以避免污染全局命名空间。

当使用IIFE模式时，也可以传递参数给它，与之前传递参数的方式一样。例如，可以这样做：

```

var i = 10;
(function(index) {
    //函数的实现代码...
})(i);

```

这个例子中定义了变量i，然后传递给IIFE。在函数内执行任意操作，需要通过index参数来使用i的值。

当需要使用事件处理器处理外部的变量时，经常使用IIFE模式。思考以下页面，包含如下代码：

```
<button id="button-1">Button 1</button>
<button id="button-2">Button 2</button>
<button id="button-3">Button 3</button>
```

我们要做的是为每个元素添加处理器，并弹出其索引（1 关于button-1，2关于button-2，3 关于button-3）。一种可能的实现代码如下：

```
for (var i = 1; i <= 3; i++) {
    document.getElementById('button-' + i).addEventListener(
        'click',
        function() { alert(i); }
    );
}
```

不幸的是，这段代码无法按照预期工作。无论怎么点击按钮，页面只会展示3。原因是在执行回调函数的时候，for循环结束，通过闭包访问的变量*i*值是3。

要解决这个问题，可以使用IIFE：

```
for (var i = 1; i <= 3; i++) {
    (function(index) {
        document.getElementById('button-' + index).addEventListener(
            'click',
            function() { alert(index); });
    })(i);
}
```

此代码为三个for循环迭代创建了新的闭包。因此，每个函数都会保留自己的*index*参数的值，轮流设置为*i*变量的值。

正如我们看到的，此设计模式十分有用，而且可以确信大家在以后的代码中会经常用到它。

A.3 总结

Summary

JavaScript是Web中广泛使用的语言，但是许多开发者并没有深入去使用这门语言。本书的附录里介绍了高效jQuery开发必备的知识概念。

如果有OO编程的背景，思考Object实例作为无须集合的键/值对容器，其实已经远离了OO编程的概念，但是它对于编写复杂的JavaScript代码是十分重要的概念。

函数作为JavaScript的一等公民，可以像其他对象类型一样声明和引用。也可以使用函数声明或函数表达式，在变量或属性里存储函数，甚至作为参数传递给其他函数作为回调函数。

函数上下文（function context）描述了函数调用期间*this*引用的对象。虽然函数可以像方法一样通过属性来设置函数上下文，但是函数不能作为单个对象方法声明。调用方式（通常由

◀ 462

调用者明确控制)用于确定函数上下文。

我们也学习了如何声明函数,以及它的环境如何组成闭包,允许函数后面调用来访问这些以及变成闭包一部分的局部变量。

最后,我们讨论了JavaScript的模式IIFE。它允许创建私有变量和函数,而且可以避免污染全局命名空间。它在处理事件回调的时候非常有用,也许需要创建闭包来确保使用正确的变量。

463 全面掌握了这些知识以后,就可以准备来迎接使用jQuery编写高性能JavaScript代码的挑战了!

修改后的代码现在显示在图A-25中来表示字符串“www.jquery.com”。

我们使用jQuery的\$.ajax()方法来获取数据,并把它放入到“#content”元素中。我们使用\$.ajax()方法来获取数据,并把它放入到“#content”元素中。

在结束附录内容之前,我想向大家介绍最后一个概念。在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。

A.2.5 立即调用的函数表达式 IIFE

IIFE是一种JavaScript设计模式,它用来创建私有变量和函数。然后,立即调用这个函数。在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。

464 立即调用的函数表达式(IIFE)是一种JavaScript设计模式,它用来创建私有变量和函数。然后,立即调用这个函数。在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。

立即调用的函数表达式(IIFE)是一种JavaScript设计模式,它用来创建私有变量和函数。然后,立即调用这个函数。在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。

在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。然后,立即调用这个函数。在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。

在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。然后,立即调用这个函数。在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。

在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。然后,立即调用这个函数。在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。

在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。然后,立即调用这个函数。在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。

在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。然后,立即调用这个函数。在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。

在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。然后,立即调用这个函数。在图A-25中,我们使用了一个立即调用的函数表达式(IIFE)来创建私有变量和函数。

索引

Index

Symbols

^ (caret character) 34
: (colon) 37, 47
. (dot operator) 450
\$() function 17–18, 23–24, 31
[] square brackets 34, 450
* (All selector) 27–29
(ID selector) 30
\$ alias 12, 15, 18, 225,
326–327
+ (Adjacent sibling
combinator) 33
> (child combinator) 33
~ (General sibling
combinator) 33

A

accepts option 292
add() method 67
addBack() method 77
addClass() method 68, 76,
101
addEventListener()
method 143
after() method 118, 122
Ajax (Asynchronous JavaScript
and XML)
Ajax events 295–298
\$.ajax() utility
function 289–293
\$.ajaxPrefilter() utility
function 298–299
\$.ajaxTransport() utility
function 299–300

contact form example
accessibility 313
effects and
animation 311–312
field validation 307–309
handler for processing
the request 309–311
HTML markup 304–305
overview 302–304
PHP backend 305–307
creating XHR object
261–263
GET requests
dynamically loading
scripts 281–283
\$.get() utility function
278–280
overview 276–278
history of 260–261
loading content using
HTML fragments
271–275
load() method 267–269
serialize() method
269–271
overview 261
POST requests 276–278,
283–284
receiving response 265–266
sending requests 264
setting request
defaults 294–295
tracking progress 265
\$.ajax() utility function
289–293, 308
ajaxComplete event 295–296
ajaxError event 295–296, 298

\$.ajaxPrefilter() utility
function 298–299
ajaxSend event 295
\$.ajaxSetup() utility function
294–295
ajaxStart event 295, 297
ajaxStop event 295–297
ajaxSuccess event 295
\$.ajaxTransport() utility
function 299–300
alert() function 230
All selector (*) 27–29
–allow-file-access-from-files
flag 56
alttitle property 406
always() method 381
AMD (asynchronous module
definition) 13, 418,
422
animate() method 206,
208–209, 213
:animated selector 46
animation
changing rate for
226–227
creating custom
drop animation example
210–211
puff animation example
211–212
scale animation example
209–210
disabling 226
easing functions 204–206
Effects Lab Page 198–200
fading elements in and out
200–202

animation (*continued*)

- queuing
 - adding functions to queue 215–216
 - clearing out unexecuted queued functions 220
 - delaying queued functions 220–221
 - executing queued functions 216–219
- showing and hiding elements
 - collapsible module 190–192
 - gradual animation
 - effect 193–198
 - overview 189–190
 - toggleing display state 192–193
 - simultaneous 213–215
 - sliding elements up and down 202–203
 - stopping 203–204
 - toggleing 312
- append() method 116–117
- appendTo() method 54, 121
- apply() method 332, 457
- array literal 451
- arrays
 - filtering 235–237
 - translating 237–238
- assert parameter 400
- assertions
 - deepEqual() method 397
 - definition 391
 - equal() method 394–397
 - notDeepEqual() method 397–398
 - notEqual() method 396–397
 - notPropEqual() method 397–398
 - notStrictEqual() method 396–397
 - ok() method 397–398
 - propEqual() method 397–398
 - strictEqual() method 395–397
 - throws() method 399–400
- Assistive Technologies. *See* ATs
- async option 291
- async() method 400, 402
- asynchronous code,
 - testing 400–402
- Asynchronous JavaScript and XML. *See* Ajax

asynchronous module definition. *See* AMD

ATs (Assistive Technologies) 314

attachEvent() method 148

attr() method 83–85, 89

attributes

- fetching values 83–84
- properties and 80–83
- removing 86
- selectors for 34–37
- setting values 84–86

autostart property 406

B

Backbone.js

- advantages of MV* frameworks 430–432
- collection 432–433, 440–441
- installing 437–438
- model 432, 438–440
- router 434–435
- Todos manager application
 - application view 443–446
 - collection 440–441
 - HTML markup 436–437
 - installing Backbone.js 437–438
 - model 438–440
 - overview 435–436
 - Todo model 438–440
 - Todos collection 440–441
 - Todos view 441–442

Backbone.localStorage

- adapter 436, 438

Basic Event Model 136

BDD (behavior-driven development) 389

before() method 118, 121

beforeSend option 291, 295

behavior-driven development. *See* BDD

blur event 156, 308

body element 7–8

Bower

- installing packages 428–429
- overview 425–427
- removing packages 429
- searching packages 427–428
- updating packages 429

\$.browser property 225

browsers

- developer tools 277

DOM Level 0 Event Model

- event bubbling 140–142
- Event object 139–140
- overview 136–139
- preventing default actions 142–143
- preventing event propagation 142

DOM Level 2 Event Model

- creating event handlers 143–145
- event bubbling 145–148
- overview 143
- event models overview 136
- Internet Explorer Model 148–149

:button selector 42

C

cache option 291

call() method 457

callback hell 358, 361

callback, function as 454–455

cancelBubble property 142

capture phase 145

caret character (^) 34

Cascading Style Sheets. *See* CSS

CDN (content delivery network) 11–12

chainability

- for plugins 337
- testing 408
- using with methods 16–17

change event 156

:checkbox selector 43

:checked selector 43

Child filters 39–42

children() method 63–64, 414

Chrome Developer Tools 29, 182

class attribute 100

Class selector 30–31, 414–415

classes, CSS

- adding and removing 100–104
- css() method 104–107

classList API 100

className property 100

clearInterval() method 349

clearQueue() method 220

CLI (command-line interface) 14, 56, 173, 426

click event 136, 156

click() method 103, 181

- clone() method 128, 176–177, 181
- closest() method 64, 66
- closures 136, 141, 459–461
- collections
 - creating from DOM element relationships 62–66
 - elements in
 - adding additional elements 66–69
 - fetching all elements as array 60
 - fetching by index 57–60
 - finding index of element 60–62
 - iterating through 233–235
 - manipulating
 - adding previous set of elements 77
 - comparing contents with selector 75
 - excluding elements from subsets 70–73
 - transforming set 73–74
 - traversing elements 74–75
 - using previous collection 76
 - Operations Lab Page 55–57
 - overview 432–433
 - size of 57
- colon (:) 37, 44
- command-line interface.
 - See CLI
- CommonJS 362
- complete event 296
- complete option 291, 295
- computed style 106
- console.log() method 29, 82
- contact form example
 - accessibility 313
 - effects and animation 311–312
 - field validation 307–309
 - handler for submit requests 309–311
 - hiding dialog box 311
 - HTML markup 304–305
 - overview 302–304
 - PHP backend 305–307
- :contains selector 44
- \$.contains() utility function 254
- content delivery network. See CDN
- Content filters 43–44
- contents option 292
- contents() method 63–64
- contentType option 291
- context parameter 17, 49, 54, 414–416
- context property 225, 291
- converters option 292
- createPseudo() function 48
- crossDomain option 292
- CSRF (cross-site request forgery) 127
- CSS (Cascading Style Sheets)
 - adding and removing classes 100–104
 - css() method 104–107
 - Modernizr and 228
 - selectors 16
- custom animations
 - drop animation example 210–211
 - puff animation example 211–212
 - scale animation example 209–210
- custom build of jQuery 14
- D**

- data option 290
- data storage, jQuery-managed 91
- data() method 92–93, 96, 333
- dataFilter option 292
- dataType option 290
- date formatter example 352
- dblclick event 156
- deepEqual() method 397–398
- default actions, preventing 142–143
- default settings for plugins 337–340
- Deferred object
 - always() method 381
 - \$.Deferred() constructor 363–364
 - determining state of 381–382
 - following progress 372–374
 - notifying about progress 371–372
 - overview 362–363
 - resolving or rejecting 364–365
 - then() method 377–381
 - using Promise object 374–377
- when() method 369–371
- define() function 422, 424
- delay() method 220
- dependency management
 - installing packages 428–429
 - overview 425–427
 - removing packages 429
 - searching packages 427–428
 - updating packages 429
- dequeue() method 216–219, 221
- Descendant selector 50
- deserialization 270
- destroy() method 336–337
- detach() method 126
- developer tools 29, 277
- dimensions, DOM element 107–112
- :disabled selector 43
- display property 189
- document ready handler 17
- DOM (Document Object Model) 4, 53–55
- DOM Level 0 Event Model
 - event bubbling 140–142
 - Event object 139–140
 - overview 136–139
 - preventing default actions 142–143
 - preventing event propagation 142
- DOM Level 2 Event Model
 - creating event handlers 143–145
 - event bubbling 145–148
 - overview 143
- done() method 365, 367
- dot operator 450
- drop animation example 210–211
- duration parameter 194
- DVD disc locator example
 - adding filters 179–182
 - controls templates 182–183
 - displaying results 183–185
 - element creation by template replication 176–178
 - filtering large data sets 174–175
 - overview 173–174
 - page markup 178–179
 - possible improvements for 186
 - removing filters 183

E

- each() method 74, 125, 234
- \$.each() utility function 234–235
- easing functions 194, 204–206
- Easing plugin 323
- ECMAScript 361–362, 418, 453
- effects 311–312
- Element selector 31–32
- elements, DOM
 - appending to DOM 53–55
 - cloning 128–129
 - creating collections from
 - relationships among elements 62–66
 - data storage for 91
 - fading in and out 200–202
 - form element values 131
 - manipulating properties of 88–91
 - moving 116–122
 - removing 126–127
 - replacing content 114–116
 - replacing element 129–131
 - showing and hiding
 - collapsible module 190–192
 - gradual animation effect 193–198
 - overview 189–190
 - tooggling display state 192–193
- sliding up and down 202–203
- styling
 - adding and removing classes 100–104
 - dimensions 107–112
 - positions and scrolling 112–114
 - setting individual styles with css() method 104–107
- wrapping and unwrapping 122–124

- elements, jQuery collection
- adding additional elements 66–69
- fetching all elements as array 60
- fetching by index 57–60
- finding index of element 60–62
- email type 304

- :empty selector 44
- empty() method 127
- :enabled selector 43
- encodeURIComponent()
 - method 245, 264, 268
- end() method 76
- :eq selector 38
- eq() method 58
- equal() method 394–397
- error event 156, 295
- error option 291, 295
- \$.error() utility function 258
- eval() function 257
- :even selector 38
- event delegation 154–155
- event module 14
- events
 - browser event models
 - overview 136
 - DOM Level 0 Event Model
 - event bubbling 140–142
 - Event object 139–140
 - overview 136–139
 - preventing default actions 142–143
 - preventing event propagation 142
 - DOM Level 2 Event Model
 - creating event handlers 143–145
 - event bubbling 145–148
 - overview 143
 - general discussion 134–136
 - Internet Explorer Model 148–149
 - jQuery Event Model
 - attaching event handlers 149–156
 - creating custom events 168–169
 - hovering over elements 166–168
 - jQuery.Event object 159–160
 - listening for event once 156
 - namespacing events 169
 - overview 149
 - removing event handlers 156–159
 - shortcut methods 165–166
 - triggering event handlers 160–164
- event.stopPropagation()
 - method 336

- exceptions, throwing 258
- expando 91
- expect() method 393, 407
- expr attribute 47
- extend() method 346
- \$.extend() utility function 242–244, 334
- extending objects 242–244

F

- F12 developer tools 29, 277
- fadeIn() method 200
- fadeOut() method 200
- fadeTo() method 201
- fadeToggle() method 201
- fading elements in and out 200–202
- fail() method 366–367
- feature detection 228, 262
- :file selector 43
- filter() method 71–72, 76, 181, 345, 416
- filterParam parameter 48
- filters
 - Child filters 39–42
 - Content filters 43–44
 - creating custom 46–49
 - definition 37–38
 - Form filters 42–43
 - optimizing performance 416–417
 - overview 44–46
 - Position filters 38–39
- find() method 63–64
- finish() method 204, 310
- Firebug plugin 29, 182, 277
- :first selector 38
- first() method 59
- :first-of-type selector 40
- \$.fn property 330, 332, 334
- focus event 156
- :focus selector 43
- focusin event 156
- focusout event 156
- forEach() method 233
- for...in loop 233
- Form filters 42–43
- forms
 - adding effects and animation 311–312
 - element values 131
 - validation 307–309
 - validation, and default actions 143

forms (*continued*)

- wrapping label-input pairs 124–126

function contexts 136

function expressions 453

functions

- as callbacks 454–455
- closures 459–461
- declaring 453–454
- IIFE 461
- overview 15, 452–453
- queuing 221
- this keyword 455–459

`$.fx.interval` property 226

`$.fx.off` property 226, 312

G

~ (General sibling combinator) 33

GET requests

- cascading dropdowns using 284–289
- dynamically loading scripts 281–283
- `$.get()` utility function 278–280
- overview 276–278
- receiving JSON data 280–281

`get()` method 58

`$.get()` utility function 278–280

`getAllResponseHeaders()` method 262

`getElementById()` function 27, 30, 414–415

`getElementsByClassName()` function 27, 30, 414

`getElementsByName()` function 31, 141, 414

`$.getJSON()` utility function 179, 280–281

`getResponseHeader()` method 262

`$.getScript()` utility function 281–283

Git 13, 426

global events 295

global option 291

global variables 179

`$.globalEval()` utility function 258

`grep()` method 185

`$.grep()` utility function 235–236

Grunt 13

`:gt` selector 38

H

`:has` selector 44

`has()` method 73, 254

`hasClass()` method 104

`hasData()` method 97–98

`:header` selector 46

headers option 292

`height()` method 107–109

`:hidden` selector 44, 46

`hide()` method 57, 189–190, 193–194

hidepassed property 406

hiding elements

- collapsible module 190–192
- gradual animation effect 193–198
- overview 189–190
- toggling display state 192–193

hierarchy selectors 32–34

`hover()` method 167

hovering over elements 166–168

`html()` method 114

HTML5 (Hypertext Markup Language 5) 100

- ID selectors and 30
- loading fragments using Ajax 271–275
- Modernizr and 228

I

ID selector 30

idempotent, defined 276

`ifModified` option 292

IIFE (Immediately-Invoked Function Expression) 229, 327, 420, 461

IIS (Internet Information Services) 266

`:image` selector 43

`$.inArray()` utility function 239

index

- fetching collection element by 57–60
- finding for collection element 60–62

`index()` method 60–61

inheritance 242

`init()` method 333–336

inline anonymous functions 455

`innerHeight()` method 111

`innerWidth()` method 111

`:input` selector 43

`insertAfter()` method 121

`insertBefore()` method 121

Internet Explorer 277, 304

- compatibility with 5, 9, 27, 101
- Event Model for 148–149
- Internet Information Services. *See* IIS

`is()` method 75

`$.isArray()` utility function 248

`isDefaultPrevented()` method 160

`$.isEmptyObject()` utility function 248

`$.isFunction()` utility function 248

`isImmediatePropagationStopped()` method 160

`isLocal` option 292

`$.isNumeric()` utility function 248

isotope plugin 324

`$.isPlainObject()` utility function 248

`isPropagationStopped()` method 160

`$.isWindow()` utility function 249

`$.isXMLDoc()` utility function 249

J

Jasmine 389

JavaScript Object Notation. *See* JSON

`jCarousel` plugin 324

`jQuery` 4–6

- document ready handler 17
- installing
 - choosing version 9–11
 - custom builds 14
 - improving performance using CDN 11–12
- `jQuery` object 15–17
- module structure 13–14
- properties 15

jQuery (*continued*)
 unobtrusive JavaScript
 overview 6–7
 script placement 7–8
 separating behavior from
 structure 7
 utility functions 15
jQuery UI 205
jQuery.Color plugin 208
jQuery.deserialize 270
jQuery.fx.interval flag 204
jQuery.fx.off flag 204
JSON (JavaScript Object
 Notation)
 defined 293
 receiving from GET requests
 280–281
 well-formed 251
json_encode() function 305
JSONP (JSON with padding)
 293
jsonp option 292
JSON.parse() function 251
jsonpCallback option 292
jsPerf 415

K

keydown event 156
keypress event 156
keyup event 156

L

:lang selector 46
:last selector 38
last() method 60
:last-child selector 40
:last-of-type selector 40
length property 57, 141
linear easing 194
lines of code. *See* LoC
load event 156
load() method 267–269, 273–
 275, 285
LoC (lines of code) 4
local events 295
:lt selector 38

M

Magnific-Popup plugin 324
\$.makeArray() utility function
 239–240

map() method 73
\$.map() utility function
 237–238
match() method 236
\$.merge() utility function
 241–242
method option 290
methods
 chaining 16–17
 defined 325
mimeType option 293
minification 10
Mocha 389
Mockjax 402
models
 overview 432
 Todos manager application
 example 438–440
Model-View-Controller pat-
 tern. *See* MVC pattern
Model-View-Presenter pattern.
 See MVP pattern
Model-View-ViewModel
 pattern. *See* MVVM
 pattern
Modernizr 228
module() method 404
moduleFilter property 406
modules
 loading with RequireJS
 421–425
 Module pattern 420–421
 object literals pattern
 419–420
 overview 418–419
 structure of 13–14
Moo Tools 3–4
mousedown event 156
mouseenter event 156, 167
mouseleave event 156, 167
mousemove event 156
mouseout event 156, 167
mouseover event 136, 156, 167
mouseup event 156
Mustache.js 433
MVC (Model-View-Controller)
 pattern 430
MVP (Model-View-Presenter)
 pattern 431
MVVM (Model-View-View-
 Model) pattern 431

N

namespaces 15
 for events 169
 jQuery/\$ 225
 for plugins 330–333
naming conventions 325–326
NaN (Not a Number) 238
nested parameters 247
Netscape Event Model 136
next() method 64, 309
nextAll() method 64
nextUntil() method 63, 65
\$.noConflict() utility function
 228–232, 327, 352
Node.js 3, 13, 426
noglobals flag 403
\$.noop() utility function 254
Not a Number. *See* NaN
:not selector 44, 46
not() method 70
notDeepEqual() method 398
notEqual() method 396–397
notify() method 371
notifyWith() method 371
notPropEqual() method 398
notrycatch flag 403–404
notStrictEqual() method
 396–397
npm 13, 320, 426
:nth-* selectors 40

O

obfuscation 10
object literals pattern 419–420
object-oriented languages.
 See OO languages
objects
 cretaing 448
 discovering type for 250–251
 extending 242–244
 functions as first-class objects
 closures 459–461
 declaring functions
 453–454
 functions as callbacks
 454–455
 IIFE 461
 overview 452–453
 this keyword 455–459
 object literals 451
 overview 448
 properties of 448–451
 testing 248–251

- objects (*continued*)
 - window properties as 451–452
- :odd selector 38
- off() method 157–158
- offset() method 112–113
- offsetParent() method 65
- ok() method 398
- on() method 150, 152–153, 182, 327
- one() method 156
- onload handler 18
- :only-child selector 40
- :only-of-type selector 40
- onreadystatechange event 263–264
- onTimeout event 263
- OO (object-oriented) languages 448
- opacity
 - fading elements in and out 200–202
 - showing and hiding elements gradually 193–198
- open() method 262, 264
- options parameter 328–329, 346–347
- originalEvent property 160
- outerHeight() method 111
- outerWidth() method 112
- overrideMimeType() method 262

P

- packages, dependency
 - installing 428–429
 - removing 429
 - searching 427–428
 - updating 429
- \$.param() utility function 245–247, 268
- parameters 327–330
- :parent selector 44
- parents() method 62, 65
- parentsUntil() method 65
- parseInt() method 105
- parsing functions 251–253
- password option 292
- :password selector 43
- performance
 - improving using CDN 11–12
 - selector
 - avoid overspecifying selectors 417–418

- avoiding Universal selector 414
- context parameter caveats 415–416
- improving Class selector 414–415
- optimizing filters 416–417
- pickadate.js 324
- placeholder attribute 87–88
- plugins
 - creating
 - destroy() method 336–337
 - init() method 333–336
 - maintaining chainability 337
 - namespacing 330–333
 - naming conventions 325–326
 - overview 325
 - parameter lists 327–330
 - providing public access to
 - default settings 337–340
 - using \$ alias 326–327
 - custom utility functions 351–352
 - defined 204
 - extending jQuery through 320
 - finding 320–321
 - overview 319–320
 - recommended plugins 324–325
 - slideshow example
 - creating plugin 344–351
 - HTML markup for 343–344
 - overview 340–343
 - using 321–324
- polyfill 228
- Position filters 38–39
- position, element 112–114
- position() method 113
- POST requests 276–278, 283–284
- \$.post() utility function 283–284, 310
- prepend() method 118
- prependTo() method 121
- prev() method 65
- prevAll() method 65
- preventDefault() method 160, 310
- prevUntil() method 65
- processData option 292

- progress() method 372, 374
- progressive enhancement 313
- promise() method 375, 382
- promises
 - creating promise object from
 - jQuery object 382
 - Deferred object
 - always() method 381
 - \$.Deferred() constructor 363–364
 - determining state of 381–382
 - executing functions 365–368
 - following progress 372–374
 - notifying about progress 371–372
 - overview 362–363
 - resolving or rejecting 364–365
 - then() method 377–381
 - when() method 369–371
 - overview 359–362
 - Promise objects 362–363, 374–377
- prop() method 88–90
- propEqual() method 398
- properties
 - attributes and 80–83
 - changing animations rate 226–227
 - disabling animations 226
 - general discussion 225–226
 - manipulating for elements 88–91
 - of objects 448–451
 - overview 15
 - \$.support property 227–228
- Prototype 3–4, 352
- \$.proxy() utility function 255–257
- pseudo-classes 37
- puff animation example 211–212

Q

- querySelectorAll() method 35, 416–417
- queue() method 216
- queuing animations
 - adding functions to queue 215–216

queuing animations (*continued*)
 clearing out unexecuted
 queued functions 220
 delaying queued functions
 220–221
 executing queued functions
 216–219

QUnit

asynchronous code testing
 400–402
configuration 405–407
grouping tests in modules
 404–405
noglobals flag 403
notrycatch flag 403–404
overview 389–392
synchronous code testing
 392–394
test suite example 407
using assertions
 deepEqual() method
 397–398
 equal() method 394–397
 notDeepEqual()
 method 397–398
 notEqual() method
 396–397
 notPropEqual()
 method 397–398
 notStrictEqual() method
 396–397
 ok() method 397–398
 propEqual() method
 397–398
 strictEqual() method
 395–397
 throws() method 399–400

R

:radio selector 43
ready event 156
ready state handler 264
ready() method 17–19
readyState property 263, 265
registry, plugin 320
regular expressions 236
reject() method 365
rejectWith() method 365
remove() method 126
removeAttribute() function 86
removeClass() method 101
removeData() method 96–97,
 336
removeProp() method 90

render() method 433
reorder property 406
replaceAll() method 130
replaceWith() method
 129–130
requests, Ajax
 custom
 \$.ajax() utility function
 289–293
 \$.ajaxPrefilter() utility
 function 298–299
 \$.ajaxTransport() utility
 function 299–300
 handling Ajax events
 295–298
 setting request defaults
 294–295
 GET requests
 cascading dropdowns
 using 284–289
 dynamically loading
 scripts 281–283
 \$.get() utility function
 278–280
 overview 276–278
 receiving JSON data
 280–281
 POST requests 276–278,
 283–284
 sending 264
 tracking progress 265
 required attribute 36, 83, 304,
 314
 requireExpects property 406
 RequireJS 421–425
 :reset selector 43
 resize event 156
 resolve() method 364, 374
 resolveWith() method 364
 response property 263
 responses, Ajax
 loading content using
 HTML fragments 271–275
 load() method 267–269
 overview 266–267
 serialize() method
 269–271
 receiving 265–266
 responseText property 263,
 265, 268
 responseType property 263
 responseXML property 263,
 265, 268
 :root selector 46
 routers 434–435

S

scale animation example
 209–210
script elements 8
scriptCharset option 292
scripts, loading dynamically
 281–283
scroll event 156
scrolling elements 112–114
scrollLeft() method 113
scrollTop property 406
scrollTop() method 113
Search Engine Result Pages.
 See SERPs
select event 156
:selected selector 43
selector property 225
selectors
 All selector (*) 27–29
 attribute selectors 34–37
 Class selector 30–31
 defined 5
 Element selector 31–32
 filters
 Child filters 39–42
 Content filters 43–44
 creating custom 46–49
 defined 37–38
 Form filters 42–43
 overview 44–46
 Position filters 38–39
 hierarchy selectors 32–34
 ID selector 30
 improving performance
 using context 49–50
 overview 26–27
 performance
 avoid overspecifying
 selectors 417–418
 avoiding Universal selector
 414
 context parameter caveats
 415
 improving Class selector
 414–415
 optimizing filters 416–417
 Selectors Lab Page 24–26
Selenium 389
send() method 262, 264
serialize() method 132,
 269–271, 285, 309
serializeArray() method 132,
 271
serializing parameter
 values 244–247

- SERPs (Search Engine Result Pages) 313
- server-side validation 302
- setInterval() method 349, 374
- setRequestHeader() method 262
- showing elements
 - collapsible module 190–192
 - gradual animation effect 193–198
 - overview 189–190
 - toggling display state 192–193
- siblings() method 65
- single responsibility principle.
 - See SRP
- single-page applications.
 - See SPAs
- Sinon.js 402
- size, collection 57
- Sizzle 417
- slice() method 72, 417
- slick plugin 323
- slideDown() method 202, 312
- slideshow example
 - creating plugin 344–351
 - HTML markup for 343–344
 - overview 340–343
- slideToggle() method 203
- slideUp() method 202, 311
- sliding elements up and down 202–203
- sort() method 240
- SPAs (single-page applications)
 - advantages of MV*
 - frameworks 430–432
 - collection 432–433, 440–441
 - defined 3
 - model 432, 438–440
 - overview 429–430
 - router 434–435
- Todos manager application
 - application view 443–446
 - collection 440–441
 - HTML markup 436–437
 - installing Backbone.js 437–438
 - model 438–440
 - overview 435–436
 - Todo model 438–440
 - Todos collection 440–441
 - Todos view 441–442
- square brackets 34, 450
- srcElement property 139

- SRP (single responsibility principle) 388
- state() method 381
- status property 263, 265
- statusCode option 292
- statusText property 263
- stop() method 203
- stopImmediatePropagation() method 160
- stopPropagation() method 142, 160–161
- strictEqual() method 395–397
- String.prototype.trim() function 232
- strings trimming 232–233
- styling elements
 - adding and removing classes 100–104
 - dimensions 107–112
 - general discussion 100
 - positions and scrolling 112–114
 - setting individual styles with css() method 104–107
- submit event 156
- :submit selector 43
- success option 291, 295
- \$.support property 227–228
- swing easing 194
- synchronous code, testing 392–394

T

- target property 87, 139
- :target selector 46
- TDD (test-driven development) 388
- template() method 433
- templates 433
- test() method 392
- test-driven development.
 - See TDD
- testId property 406
- testing
 - asynchronous code 400–402
 - grouping tests in modules 404–405
 - importance of 386–387
 - noglobals flag 403
 - notrycatch flag 403–404
 - objects 248–251
- QUnit
 - configuration 405–407
 - overview 389–392
- synchronous code 392–394
- test suite example 407
- unit testing
 - frameworks for 388–389
 - importance of 387–388
- using assertions
 - deepEqual() method 397–398
 - equal() method 394–397
 - notDeepEqual() method 397–398
 - notEqual() method 396–397
 - notPropEqual() method 397–398
 - notStrictEqual() method 396–397
 - ok() method 397–398
 - propEqual() method 397–398
 - strictEqual() method 395–397
 - throws() method 399–400
- testing objects 248–251
- testTimeout property 406
- :text selector 43
- text() method 115, 122, 309
- then() method 360, 362, 377–381
- this keyword 66, 337, 349, 455–459
- throwing exceptions 258, 399–400
- timeout option 291
- timeout property 263
- toArray() method 60, 74
- Todos manager application
 - application view 443–446
 - collection 440–441
 - HTML markup 436–437
 - installing Backbone.js 437–438
 - model 438–440
 - overview 435–436
 - Todo model 438–440
 - Todos collection 440–441
 - Todos view 441–442
- toggle() method 192–193, 196
- toggleClass() method 102–103
- Tomcat 266
- traditional option 293
- trigger() method 160–161
- triggerHandler() method 162–163, 348
- \$.trim() utility function 232

trimming strings 232–233
truthy values 234
\$.type() utility function 250
typeahead.js 324

U

Underscore.js 433, 437
Unheap 321
\$.unique() utility function
240–241
unit testing
asynchronous code 400–402
frameworks for 388–389
grouping tests in modules
404–405
importance of 387–388
noglobals flag 403
notrycatch flag 403–404
QUnit
configuration 405–407
overview 389–392
synchronous code 392–394
test suite example 407
using assertions
deepEqual() method
397–398
equal() method 394–397
notDeepEqual()
method 397–398
notEqual() method
396–397
notPropEqual() method
397–398
notStrictEqual() method
396–397
ok() method 397–398
propEqual() method
397–398
strictEqual() method
395–397
throws() method 399–400
Universal selector 414
unload event 156

unobtrusive JavaScript
overview 6–7
script placement 7–8
separating behavior from
structure 7
unwrapping elements 122–124
upload property 263
url option 290
urlConfig property 406
useCapture parameter 146
username option 292
utility functions
custom 351–352
defined 325
discovering type for values
250–251
doing nothing 254
evaluating expressions
257–258
extending objects 242–244
filtering arrays 235–237
iterating through collections
233–235
overview 15
parsing functions 251–253
prebinding function contexts
255–257
serializing parameter values
244–247
testing for containment
254–255
testing objects 248–251
throwing exceptions 258
translating arrays 237–238
trimming strings 232–233
using noConflict() function
with other libraries
228–232

V

val() method 131–133
validation
server-side vs. JavaScript 302

using Ajax 307–309
var keyword 452
variables, global 179
velocity.js 324
views
overview 433–434
Todos manager application
example 441–442
:visible selector 44, 46

W

W3C (World Wide Web
Consortium) 27
Web Storage API 436
when() method 369–371
width() method 107–109
window object 452
window.alert() method 29, 82
window.navigator.userAgent
property 391
withCredentials property 263
World Wide Web Consortium.
See W3C
wrapping elements
overview 122–124
wrapping label-input pairs of
form 124–126

X

XHR object, creating 261–263
xhr option 292
xhrFields option 292
XMLHTTP ActiveX control
261
XMLHttpRequest 260–261

Y

YUI Test 389

jQuery 实战（第三版）

Bear Bibeault • Yehuda Katz • Aurelio De Rosa

感谢jQuery，没有人再愿意记起之前糟糕的日子：

程序员手动调试浏览器兼容性，CSS选择器支持，DOM导航，以及JavaScript支持的恐怖动画效果。优雅、直观的jQuery库精美处理了这些问题，jQuery 3又新增了更多的新特性，让Web开发者工作更平滑、更高效！

《jQuery实战》（第三版）是一本学习jQuery的快速指南，关注Web项目中常见的问题。本书中，我们将学习如何遍历DOM、处理事件、执行动画、编写jQuery插件、Ajax请求及单元测试。此外，本书还提供了独特的试验页面，针对实际代码中的每个概念在线演示。第三版的扩展内容章节会告诉我们如何与其他工具和框架交互，以及如何构建新的单页Web应用。

内容包含：

- 更新到jQuery 3。
- DOM操作与事件处理。
- 动画与特效。
- 高级主题：单元测试与Promises。
- 实战例子与试验。

读者对于JavaScript编程有基本的了解。

Bear Bibeault 是《Secrets of the JavaScript Ninja》、《Ajax in Practice》及《Prototype and Scriptaculous in Action》的共同作者。**Yehuda Katz**是jQuery项目的早期代码贡献者，Ember.js项目的联合创始人。**Aurelio De Rosa**是一位全栈工程师，参与过很多开源项目，也是jQuery内容团队的成员。

上架建议：Web开发>前端开发

策划编辑：谢燕群
责任编辑：陈元玉

Free eBook
SEE INSERT

详细介绍了jQuery框架各个组件如何协同工作，并且提供了重要概念的演示代码。

——来自Dave Methvin jQuery
基金会主席 推荐序

jQuery学习最佳书籍，对jQuery框架研究最详细、最全面、最深入的书籍。

——Jonnn Resig, jQuery之父

现在已经第三版了，这也是我唯一向客户推荐的jQuery学习书籍。

——Christopher Haupt
Mobirobo公司 架构师

ISBN 978-7-5680-2035-0



9 787568 020350 >

定价：88.00元